Título: Desarrollo de una plataforma de trabajo para la investigación

Volumen: 1/1
Alumno: Víctor Blázquez Francisco

Director/Ponente: Daniel Jiménez González
Departamento: Arquitectura de computadores
Fecha: 22 de Junio del 2010

DATOS DEL PROYECTO

Título del proyecto:	Desarrollo de una plataforma de trabajo para la investigación
Nombre del estudiante:	Víctor Blázquez Francisco
Titulación:	Enginyeria Técnica en Informática de Sistemas
Creditos:	22,5
Director/Ponente:	Daniel Jiménez González
Departamento:	Arquitectura de computadores

MIEMBROS DEL TRIBUNAL *(nombre y firma)*

Presidente:	Carlos Álvarez Martínez
Vocal:	Pere Pau Vázquez Alcocer
Secretario:	Daniel Jiménez González

CALIFICACIÓN

Calificación numérica:

Calificación descriptiva:

Fecha:

CiFPGA: Core in FPGA

Víctor Blázquez Francisco

22 de Junio del 2010

Índice general

1. Introducción

Software Open Source? Seguro que has oído hablar de ello, seguro que conoces grandes proyectos, *Apache, Linux, X.Org, perl, Firefox, java, etc*, pero... has oído hablar de *Hardware Open Source*? Bien, como su nombre indica, es una plataforma hardware con el sistema de código abierto, es decir, no sólo tienes todas las especificaciones de la placa, sino que también tienes una descripción del circuito, lo que viene a significar *Open Source*. Así, de esta manera, puedes conocer todo el funcionamiento interno de tu placa, adaptar tus programas de acuerdo con el hardware que vas a usar y que éste funcione de la manera más óptima y eficiente.

A nivel hardware existen múltiples diseños en multitud de temas, quizás el más complejo que se puede diseñar a día de hoy es un procesador. En la actualidad, existen varios proyectos sobre microprocesadores de *Hardware Open Source*, pero aún habiendo un gran catálogo de ellos, no se puede decir que todos sean procesadores como los que tenemos en casa. Algunos tienen limitaciones en cuanto a su desarrollo ya que no han llegado a completarse con todas las funcionalidades a las que estamos acostumbrados y/o conocemos. Los más completos y que merecen ser destacados por su magnitud e importancia a día de hoy son *Leon* y *OpenSPARC*.

En 2005, el microprocesador *UltraSPARC T1* fue liberado bajo el nombre de *OpenSPARC* por parte de *Sun Microsystems* bajo la *licencia pública general, GPLv2*. *OpenSPARC* es un procesador completo escrito en código *VHDL* y con la completa descripción del circuito. Al ser *Open Source* permite ver y modificar el código, lo cual es una ventaja ya que permite tener un modelo real y funcional, donde no sólo los desarrolladores de *Sun* pueden trabajar con él, sino todo el mundo, y así de esta manera no sólo mejorarlo, sino que se pueden ampliar las capacidades de las que éste dispone y ofrecerlas a la comunidad. Actualmente *OpenSPARC* lleva 4 años en abierto y con dos versiones del procesador, *OpenSPARC T1 y T2*. Además cuenta con el respaldo de *Sun*, que es una gran compañía, y también de una gran comunidad que se ha ido formando a lo largo de estos años, no sólo ofreciendo soporte, sino mejorando sus características.

Una vez se tiene un microprocesador, ya sea en forma de código o *hardware físico*, una de las primeras cosas que se hace es probarlo para ver sus resultados, el funcionamiento, posibilidades que ofrece, etc. Para poder probarlo hace falta tener una placa base, siempre y cuando hablemos de *hardware físico*, o de una *FPGA* (*Field Programmable Gate Array*), en el caso de que hablemos de un procesador escrito en forma de código. Mediante una *FPGA* existe la posibilidad de que, modificando el código, se pueda probar el resultado del microprocesador sin la necesidad de tener que construir otra vez el microprocesador físicamente, lo que supone un ahorro en muchos sentidos, no sólo refiriéndonos al dinero. *Altera* y *Xilinx* son los dos principales proveedores en este campo. *Xilinx* ofrece una familia de *FPGAs* totalmente compatible y diseñadas específicamente para *OpenSPARC*, además de proveer herramientas específicas para trabajar con ambos proyectos. La familia que se va a utilizar es la *Virtex-5*.

Una *Virtex-5* en conjunción con su placa, *ML505*, cumple muy bien el propósito de poder recibir unos *inputs* y a su vez mostrar unos *outputs*. Como resultado de su unión junto con *OpenSPARC* se obtiene un modelo de microprocesador completo. Con el que se puede trabajar tal y como se verá más adelante, y en el que finalmente se arrancará un *Sistema Operativo*.

La existencia de todas estas herramientas permiten avanzar en el campo de la investigación de microprocesadores de una manera más rápida y efectiva. Anteriormente, una vez se había diseñado el microprocesador, repasados sus cálculos y se estaba completamente seguro de que todo funcionaba tal y como se esperaba, se procedía a crear un prototipo. Hoy en día este prototipo se crea en forma de código, con el consecuente ahorro de materiales, además de ofrecer la posibilidad de testearlo. En caso de encontrar errores, son corregidos de una manera mucho más ágil que con un procesador físico.

A día de hoy, no sólo la investigación por parte de *SUN*, *AMD*, *Intel*, están interesados en todo esto, la docencia también lo está, ya que permite dar clases no sólo a nivel teórico, sino también práctico. Una persona que se dedique a desarrollar no debería preocuparse por cómo compilar, qué herramientas utilizar, cómo se utilizan, etc. El programador debería centrarse en programar, por lo que tener automatizados todos estos procesos es algo muy interesante.

En este trabajo se muestra una solución que consiste en una plataforma que automatiza la compilación, ejecución, obtención y mostrado de resultados de una manera ágil y eficiente. De esta manera, una persona sólo debería preocuparse por saber cómo funciona el procesador, encargarse de modificarlo, adaptarlo o lo que se desee y comprobar si los resultados son satisfactorios o no de acuerdo a los cambios pertinentes.

Para simplificar la plataforma, se ha desarrollado un portal web donde se recogen los códigos de los usuarios y donde se ofrecen los resultados. Internamente la plataforma es la encargada de compilar todo aquello necesario, arrancar la *FPGA*, probar el código del usuario y guardar los resultados, para que a posteriori sean mostrados al usuario, junto con un histórico de resultados de todo lo que se ha ido haciendo.

Los resultados del proyecto han sido la obtención de una plataforma que permite al alumno/investigador preocuparse sólo por el código y sus resultados sin tener que pasar por el tedioso proceso de tener que aprender cómo funciona la conexión host-placa, cómo se compila y todo lo que son los pasos intermedios como problemas a la hora de compilar

A continuación se describen brevemente los resultados de este trabajo:

1. *Scripts*: *scripts* de compilación para poder compilar el código de *OpenSPARC*, enviarlo a la placa y mostrar los resultados conjuntos de todo ello.

2. Portal web: permite tener una página web personalizada desde donde se pueden enviar los códigos y comprobar sus resultados, añadiendo un histórico sobre las compilaciones anteriores.

3. Configuración de la placa: ofrece información específica sobre cómo conectar la *FPGA* al *host* anfitrión y configurar la conexión entre ambos.

4. *Cross-compiler*: permite la compilación de código en una máquina *x86* contra una máquina con arquitectura *SPARC*.

5. Sistema operativo: modificación del sistema operativo que trae por defecto el proyecto *OpenSPARC*, añadiéndole funcionalidades extras.

6. *Servidor FTP* en el *host*: permite la obtención y envío de ficheros mediante el servidor *FTP*, para cuando se arranca la *FPGA* con un sistema operativo y así poder enviar y recibir datos.

7. Imagen *ISO*: ofrece un *LiveCD* con *GNU/Linux* donde se encuentra todo esto anteriormente mencionado e integrado para que funcione correctamente, y resolviendo los problemas básicos que se pueden generar durante la compilación y ejecución del código.

El *LiveCD* que se ofrece como resultado, integra todo lo descrito anteriormente, a excepción de las herramientas de *Xilinx* debido a un problema de licencia y de espacio, ya que un *DVD* no ofrece suficiente capacidad.

Este documento está organizado de la siguiente manera:

1. FPGA - Virtex-5: hace un poco de historia sobre las *FPGAs* y finalmente se centra en la *Virtex-5*.

2. OpenSPARC: trata sobre el procesador *OpenSPARC* hablando sobre su historia, definiendo parte de sus características y cómo se debería generar el *bitstream*.

3. Entorno de trabajo: trata sobre todo el material relacionado para poder reproducir fielmente el proyecto.

4. Plataforma de investigación:

   a) Scripts de compilación y ejecución: explica los *scripts* utilizados para la generación del *netlist* y del *bitstream* y finalmente la ejecución.

   b) Arranque de un sistema operativo: trata sobre cómo arrancar un sistema operativo sobre la *FPGA*, cómo probar un código específico y cómo modificar el sistema operativo.

   c) Cross-compiler: generación de un cross-compiler para poder compilar programas desde una arquitectura *x86* a arquitectura *SPARC*.

5. Portal web: página web que permite enviar y obtener resultados sobre los códigos enviados sobre *OpenSPARC* para ser testeados.

6. LiveCD: trata sobre cómo generar un *CD botable* que permite tener todas las herramientas utilizadas durante este proyecto en un *CD*.

7. Planificación y costes: trata sobre cómo se ha organizado este proyecto y lo que costaría reproducirlo.

8. Conclusiones: valoración sobre los resultados obtenidos en este proyecto.

2. FGPA - Virtex-5

Durante este capítulo se hará un poco de historia sobre las *FPGAs*, para qué sirve, en qué se diferencia de los *CPLDs* y concretamente se hablará sobre la *Virtex-5*.

Una *FPGA* [6], *Field Programmable Gate Array*, es una evolución natural de un *CPLD* [5], en la figura 2.1 se muestra un *CPLD* de la marca *Altera*. Las *FPGAs* nacieron en 1984 de la mano de Ross Freeman y Bernard Vonderschmitt, actualmente cofundadores de la marca *Xilinx*. Hoy en día, *Xilinx* [15] es uno de los grandes referentes en el mercado de *FPGAs* y *CPLDs* junto con su gran rival *Altera*.

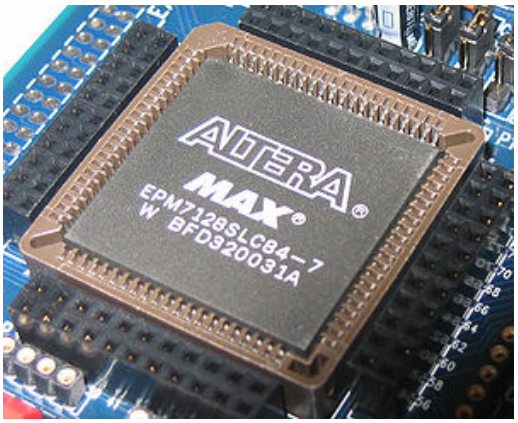

Figura 2.1: *CPLD de Altera*

Una *FPGA*, al igual que un *CPLD*, es un conjunto de bloques o elementos lógicos programables, sólo que la *FPGA* permite reprogramar un número más elevado de puertas lógicas, tiene una estructura mucho más flexible y algunas funciones de alto nivel embebidas en la propia matriz, lo que permite crear estructuras mucho más complejas y así, de esta manera poder generar circuitos lógicos mucho más grandes y elaborados. En la figura 2.2 se muestra una *FPGA* junto con su placa, una *SPARTAN*.

Mediante interconexiones de los bloques lógicos de una *FPGA*, se puede conseguir que emule cualquier función lógica, incluso hasta una CPU, tal y como se verá más adelante. La tendencia actual, consiste en ampliar las funcionalidades de una *FPGA* añadiendo periféricos y microprocesadores, para así formar una tecnología híbrida, una parte reprogramable y otra estática. Como ejemplo de esta tecnología híbrida esta la familia *Virtex-5*, perteneciente a *Xilinx*.

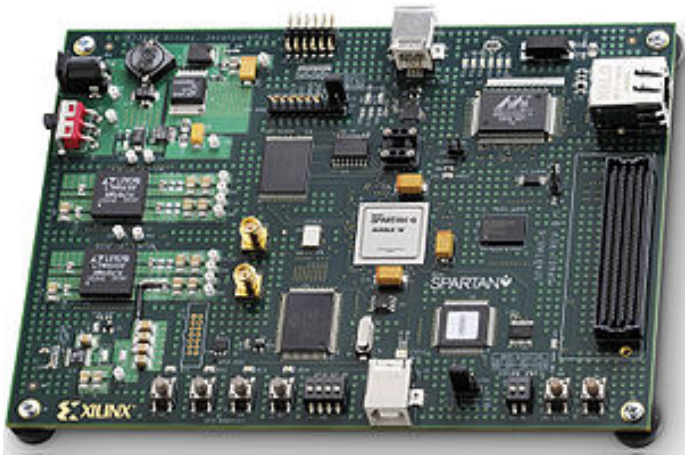

Figura 2.2: *Ejemplo de una FPGA y su placa*

Dentro de la familia Virtex-5, encontramos la placa *ML505* [2], la cuál tiene unas características muy importantes entre las que destacamos principalmente:

- Amplio soporte de periféricos tales como *PS/2*, *USB*, *Ethernet*, *Fibra Optica*, etc.
- Dispositivos de salida gráfica externa *VGA*, *HDVI*.
- Memória RAM, por defecto de 256Mb, ampliable.
- *Puerto Serie* y *JTAG* para poder reprogramar la *FPGA*.
- Jumpers de configuración internos.
- Display para mostrar información sin necesidad de una salida gráfica.
- Soporte para memória física, en formato *Compact Flash* o *SATA*.

En la figura 2.3, se puede ver la *FPGA Virtex-5*, en concreto el modelo *XC5VLX50T*, junto con la placa *ML505*. En ella se pueden ver las características antes mencionadas, entre otras tantas.

Para programar una *FPGA* se hace diseñando las interconexiones entre los bloques lógicos que ésta posee. Un programador puede desarrollar cualquier función lógica, que tras un proceso de síntesis y posterior inserción en la *FPGA*, ésta desempeñará.

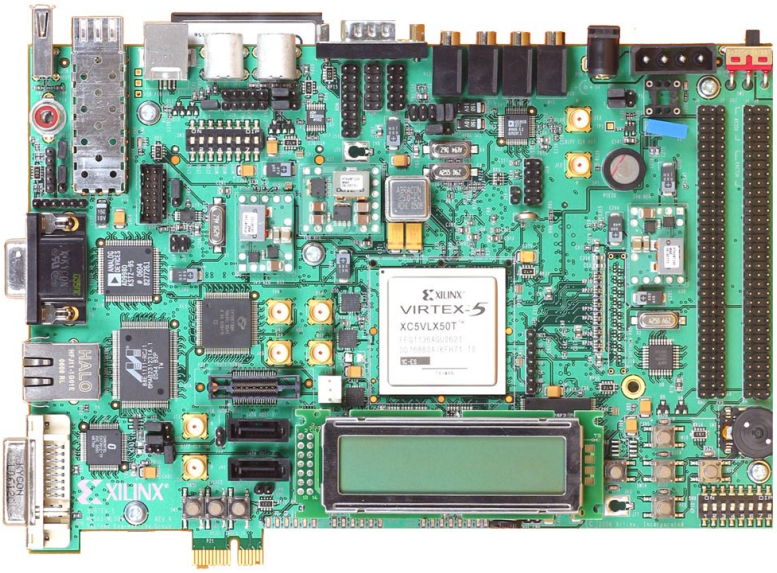

Figura 2.3: *Virtex-5 Ml505*

Para poder diseñar este código se suele utilizar un *IDE*. Un *IDE* proporciona información gráfica de cómo se encuentran interconectados los bloques y así permite una mayor agilidad a la hora de desarrollar. Para poder diseñar el conexionado de bloques se hace mediante código *VHDL* o *Verilog* principalmente.

Tras un proceso de síntesis se genera un *bitstream*; que es un fichero que contiene la configuración que luego será insertada en la *FPGA*.

Finalmente, cuando se arranque, se podrán comprobar los resultados del programa introducido. Para introducir el *bitstream*, hay varias maneras, desde reprogramar una *EPROM* a las basadas en *RAM*, que una vez se desconecte de la alimentación pierden todos los datos almacenados.

Cómo ya se ha comentado, una *FPGA* permite ser programada para emular cualquier función lógica que se pueda imaginar. Una de las más complejas que se puede imaginar a día de hoy, es el diseño de una *CPU*. *OpenSPARC* es un diseño abierto de una *CPU* real, especialmente diseñada y adaptada para funcionar sobre una *Virtex-5 ML505*. Durante el siguiente capítulo se hablará sobre este microprocesador en forma de código.

3. OpenSPARC

OpenSPARC T1 es un procesador completo, es decir que consta de todas las características de las que dispone cualquier procesador real y con sus correspondientes módulos. Una de las principales características de *OpenSPARC* [13] es que a parte de ser libre, es decir que se dispone de todas sus especificaciones, es que está escrito en *VHDL*, es decir en formato de código. Esto permite que pueda ser desarrollado y ampliado por cualquier persona que tenga interés en el tema.

Este microprocesador está basado en *UltraSPARC T1*, figura 3.1, y que funciona sobre *RISC*, *Reduced Instruction Set Computer*. *OpenSPARC* vio la luz en Marzo del 2006. Aunque *UltraSPARC* comenzó a comercializarse a partir de diciembre del 2005. Por lo que se puede decir que *OpenSPARC* tiene una edad aproximada de 5 años y no sólo con la comunidad de *SUN Microsystems* detrás suyo, sino que también tiene una comunidad formada a su alrededor, gracias a la liberación de su código.

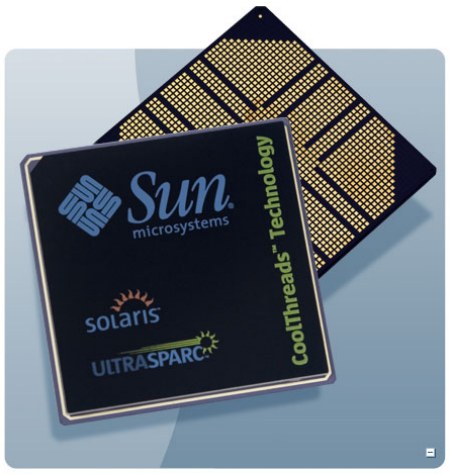

Figura 3.1: *Chip UltraSPARC*

Las principales características del microprocesador completo *OpenSPARC* [3] son:

- Tiene 8 núcleos con 4 hilos por núcleo haciendo un total de 32 hilos
- Ésta basado en una arquitectura *SPARC*, *Scalable Processor ARChitecture*.
- Tiene una cache de nivel 2 integrada en el propio chip
- Tiene un espacio de direcciones virtual de 48 bits y 40 bits físico
- Dispone de 16 kbytes de cache de instrucciones y 8kbytes de datos por *core*.
- Tiene una interfaz J-Bus

En la figura 3.2 se pueden ver a grandes rasgos los principales bloques de los que posee *OpenSPARC*.

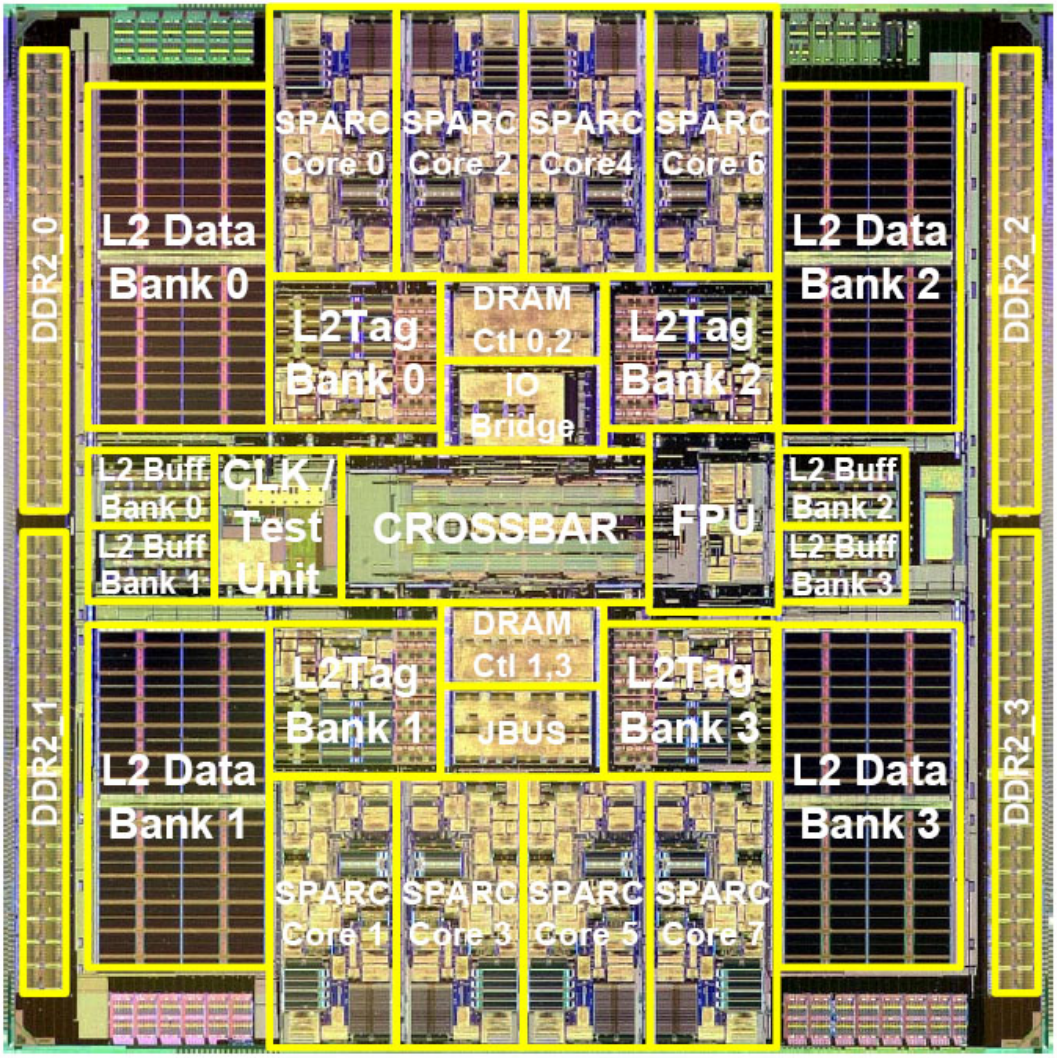

Figura 3.2: *Diseño de bloques de OpenSPARC*

Enfatizar que cada núcleo utiliza una arquitectura *SPARC* de *64 bits* la cual es completamente compatible con su predecesora de *32 bits* [10]. Y aunque *OpenSPARC* venga de serie con 8 núcleos y 4 hilos por cada uno, se puede compilar de tal manera que se tenga 1 núcleo y un hilo por procesador, lo cual nos muestra parte de su gran versatilidad para adaptarse y adaptarlo a nuestras necesidades.

OpenSPARC permite añadir coprocesadores de todo tipo [11], aceleración de coma flotante, enrutado de redes, módulos criptográficos, *streaming* de vídeo, interfaces I/O, etc.

La figura 3.3 muestra un diagrama de bloques del procesador *OpenSPARC T1*, donde se pueden apreciar las diferentes interfaces y componentes integrados en el chip. *OpenSPARC* ofrece una documentación detallada y precisa sobre el funcionamiento y las características que éste posee [1].

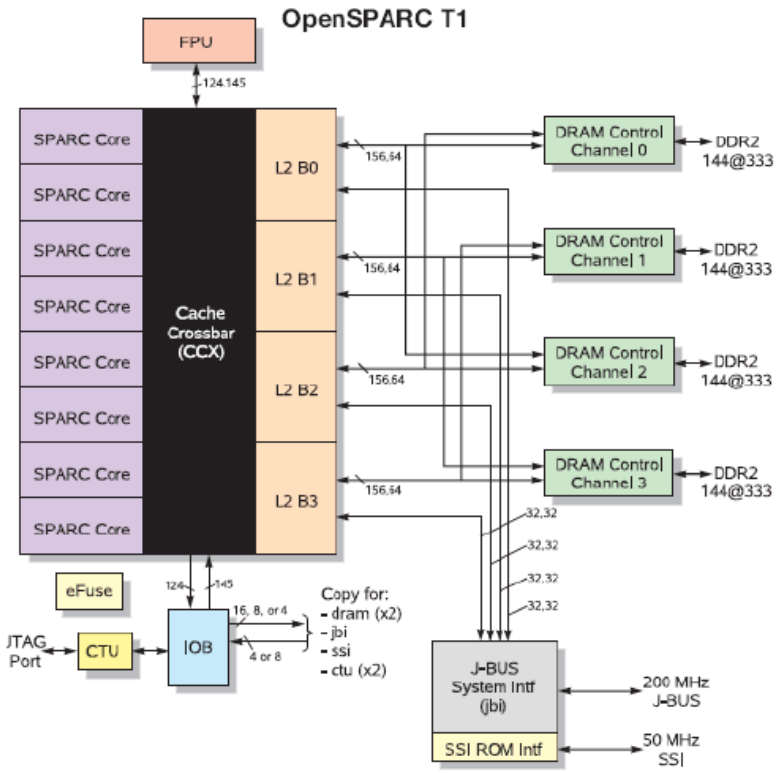

Figura 3.3: *Diagrama de bloques*

Y quizás el detalle más importante en cuanto al proyecto *OpenSPARC*, es que está totalmente soportado por *Xilinx*. Lo que ofrece un gran abanico de posibilidades, ya que se dispone de una *FPGA* que soporta nativamente *OpenSPARC*, lo que es una facilidad añadida, ya que sabemos que tenemos una base en la que funciona correctamente y a partir de la cual se puede empezar a trabajar.

Para sintetizar *OpenSPARC* es necesario generar un *bitstream*. Un *bitstream* es un fichero binario que sirve para configurar las funciones y conexiones que se programan en la *FPGA*. Para poder generar este fichero binario, se utiliza el código *OpenSPARC* que está escrito en lenguaje *VHDL* principalmente y que se encuentra disponible en su página web [8].

La generación del *bitstream* consta de dos pasos principales:

1. Generar *netlist*: consiste en generar el *netlist*, que es la especificación de los componentes de los que consta la electrónica y de cómo se intercomunican entre sí.

2. Generar *bitstream*: una vez se tiene la información básica de cómo está todo interconectado se procede a generar el *bitstream* necesario para la *Virtex-5*, este proceso es más conocido como sintetizar. Cada software de compilación funciona bajo la *FPGA* sobre la que ha sido diseñado, ya que el sistema y el hardware que utilizan no es libre y no se sabe como funciona realmente.

4. Entorno de trabajo

Para la creación de la plataforma se ha utilizado una máquina con las siguientes características hardware:

- Intel Core 2 Duo, 2,66GHz
- 4Gb de memoria RAM
- Disco duro de 500Gb
- Diversos conectores usb para la conexión *host-placa*.
- Adaptador *USB-RS232*.

Para el entorno de desarrollo se han utilizado dos Sistemas Operativos diferentes:

- Debian, con un kernel 2.6.30
- Ubuntu, con un kernel 2.6.31.

No existe una gran diferencia entre estas dos distribuciones. El usar dos distribuciones diferentes viene dado porque el proyecto se empezó con *Debian* y llegados a un punto avanzado del proyecto, se decidió crear un *LiveCD* y se consideró que la distribución más conveniente era *Ubuntu*. Para el siguiente paso, era necesario testear y comprobar que todo lo realizado funcionaba correctamente en la nueva distribución, lo que funcionó perfectamente ya que las dos distribuciones son muy parecidas.

Cualquier paso del que se hable de aquí en adelante sobre el sistema anfitrión, está preparado para funcionar bajo cualquiera de las dos distribuciones anteriormente mencionadas.

Para generar los diversos ejecutables han hecho falta varios tipos de compiladores:

- gcc: compilador nativo de *GNU/Linux*, utilizado para compilar el *cross-compiler*.
- cross-compiler: compilador que permite compilar programas para la arquitectura *SPARC* desde una máquina con arquitectura *x86*.
- xst: es una de las herramientas proporcionadas por *Xilinx* para poder compilar el código *VHDL* en el cual está escrito *OpenSPARC* y obtener un bitstream.
- rxil: herramienta para sintetizar el netlist de *OpenSPARC*.

También es necesaria una *FPGA*, en este caso una *Virtex-5 V5LX110T* con una placa *ML505*, tal y como se puede ver la figura 4.1. También se puede observar un conversor de *JTAG* a *usb* y finalmente un cable *RS-232*.

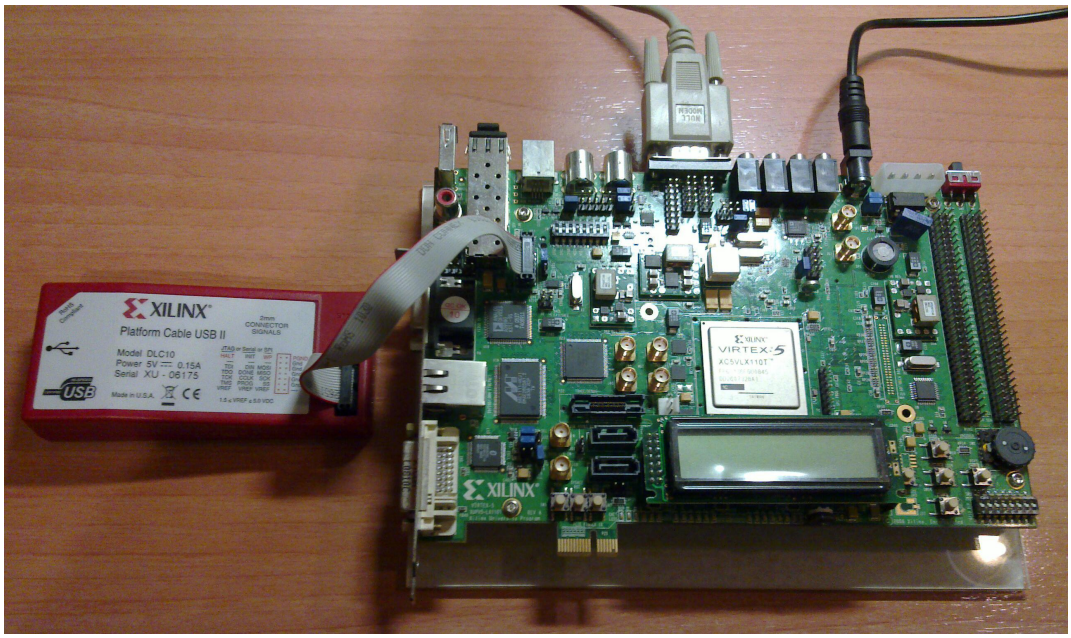

Figura 4.1: *Imagen de la FPGA*

Para poder conectar la placa al host anfitrión, es necesario un conversor *USB* ya que el ordenador no dispone de puerto serie o *RS-232*. En la figura 4.2 se puede ver un conversor de *USB* a *RS-232*. Por defecto, *GNU/Linux* no es capaz de reconocerlo, por lo que es necesario instalar los drivers.

Primeramente es necesario instalar el paquete *libusb-dev*, que contiene las cabeceras de la librería *libusb*, tal y como se observa la línea 1. Seguidamente descargar y descomprimir el driver [9]. Finalmente en la línea 5 se compila para acto seguido, cargarlo como variable de entorno en el sistema.

Figura 4.2: *Conversor USB a RS-232*

```
1 aptitude install libusb-dev
2 wget http://git.zerfleddert.de/cgi-bin/gitweb.cgi/usb-
     driver?a=snapshot;h=HEAD;sf=tgz
3 tar -zxf usb-driver-HEAD.tar.gz
4 cd usb-drivers
5 make
6 export LD_PRELOAD=/ruta/de/libusb-driver/libusb-driver.so
```

En la última del fragmento anterior se muestra cómo cargar el *driver* manualmente. Pero no es recomendable ya que genera problemas con otros programas instalados. Por lo que será cargado cada vez que se necesite y esto se hará de forma automática en los *scripts* de compilación, mirar apartado 5.1.

En el fragmento que viene a continuación, nos aseguramos de que el driver siempre funcione, ya que a veces *udev* necesita más información [14], como es el actual caso. Por lo que haría falta añadir alguna regla a *udev*, la cual viene preparada por *Xilinx* y que hace falta adaptarla. En la primera línea se copia en */etc/udev/rules.d/* a la vez que se modifica el contenido de esta ya que a *udev* no le gusta en mayúsculas. Seguidamente se copian ficheros que necesitará y finalmente se reinicia el demonio.

```
1 sudo sed $Xilinx/ISE/bin/lin/xusbdfwu.rules -e 's:TEMPNODE
     :tempnode:g' > /etc/udev/rules.d/xusbdfwu.rules
2 sudo cp $Xilinx/ISE/bin/lin/xusb*.hex /usr/share/
3 sudo /etc/init.d/udev restart
```

En la figura 4.3 se puede ver como el demonio de *udev* ha detectado correctamente el *usb*, después de ejecutar el comando *dmesg |grep usb*.

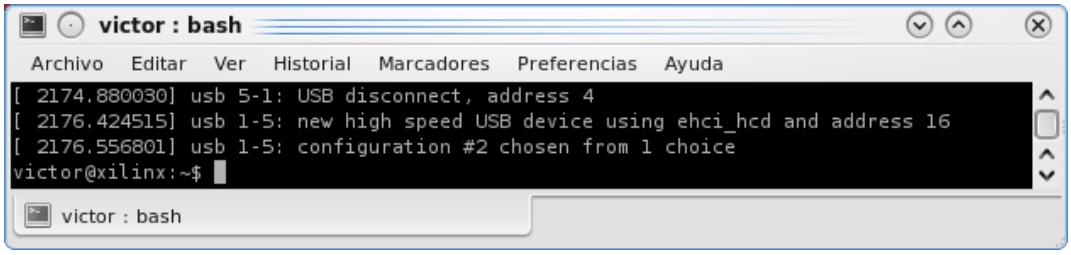

Figura 4.3: *Detección del USB por parte del Host anfitrión*

Para ver los resultados de la salida serie se utiliza durante el proyecto *minicom* o *cutecom*. El primero funciona bajo línea de comandos, mientras que el segundo funciona en un entorno gráfico.

Finalmente es necesario el código de *OpenSPARC*, que se puede descargar desde su web [8]. El cual consta del código y documentación acerca de funcionamiento, arquitectura y otros documentos interesantes.

5. Plataforma de automatización

A lo largo de este capítulo se detallará cómo conseguir una plataforma de automatización para la investigación/docencia con *Virtex-5* y *OpenSPARC*. Se podrá observar cómo compilar *OpenSPARC* e insertarlo en la *FPGA* para finalmente poder observar resultados, ya sea arrancando un sistema operativo o enviando un código especifico de test, creado mediante el *cross-compiler* que se explicará también en este capítulo.

5.1. Scripts de compilación y ejecución

A lo largo de esta sección se detallará cómo conseguir un *script* de compilación para *Virtex-5* y *OpenSPARC*. En ellos se podrá observar cómo compilar *OpenSPARC* e insertarlo en la *FPGA* para finalmente poder observar resultados enviando un código específico de test.

5.1.1. Compilación de OpenSPARC

Para el proceso de síntesis e inserción en la *FPGA* es necesario definir unas variables relativas al proyecto, tales como dónde se encuentra el código de *OpenSPARC*, variables que define el software de compilación de *Xilinx*, etc.

Con las siguientes líneas se cargan variables de entorno que necesita *Xilinx*, para poder compilar y/o ejecutar su *IDE*. Estos dos *scripts* se encuentran en el directorio por defecto de la instalación del *EDK* y del *ISE* de *Xilinx*, tal y como se puede observar a continuación:

```
1 source $Xilinx/EDK/settings32.sh
2 source $Xilinx/ISE/settings32.sh
```

Seguidamente se detallan variables que necesita el compilador de *Xilinx* para saber dónde se encuentra el proyecto a compilar, especificado por las variables de las líneas 1 a 3, dónde se encuentran los binarios del sistema, líneas 4 y 5. Las siguientes 4 líneas marcan dónde se ubicará el fichero de *log* y en caso de no existir se creará. Las siguientes 3 líneas especifican variables relativas al código descargado de *OpenSPARC*. Finalmente la última línea actualiza el *$PATH* del sistema.

```
1  export DV_ROOT=$HOME/OpenSPARC/
2  export MODEL_DIR=$HOME/OpenSPARC/
3  export TRE_ENTRY=/
4  export CC_BIN=/usr/bin
5  export PERL_CMD="/usr/bin/perl"
6  if [ -f $HOME/log_opensparc ]; then
7          touch $HOME/log_opensparc
8  fi
9  export TRE_LOG=$HOME/log_opensparc
10 export TRE_SEARCH="$DV_ROOT/tools/env/tools.iver"
11 export ENVDIR=$DV_ROOT/tools/env
12 export PERL_MODULE_BASE=$DV_ROOT/tools/perlmod
13 export PATH=".:$XILINX/EDK/gnu/microblaze/lin/bin/:
     $DV_ROOT/tools/bin:$CC_BIN/:$PATH"
```

Para conseguir un binario con un *OpenSPARC* reducido, es necesario editar el fichero *$DV_ROOT/design/sys/iop/include/xst_defines.h*. Tal y como se detalla en el *DV_Guide* [12], que es la documentación ofrecida por *OpenSPARC*, se explica cómo reducir el tamaño del binario a generar. Además estas opciones se pueden usar para adaptarlo a otras necesidades. Como ejemplo se podría modificarlo a sólo un *thread*, o con 8 entradas de *TLB* en lugar de las 64 por defecto, anular módulos como el *SPU*, módulo de aceleración criptográfica, etc. Añadir que las herramientas de *Xilinx* no permiten hacerlo de otra manera.

Como se puede ver seguidamente se definen las opciones para generar *OpenSPARC* con un tamaño reducido. Se anula el módulo de la *SPU*, se limita a un solo *thread* y se reduce el *TLB* a 8 entradas.

```
1 `define FPGA_SYN
2 `define FPGA_SYN_NO_SPU
3 `define FPGA_SYN_1THREAD
4 `define FPGA_SYN_8TLB
```

El siguiente paso es comenzar a sintetizar. Esto se hace mediante la herramienta *rxil*, indicándole el tipo de dispositivo que vamos a utilizar, el dispositivo *XC5VLX110T*, que es el correspondiente a la *Virtex-5 ML505*. Los bloques a sintetizar son los siguientes:

1. *sparc*: es la unidad de procesamiento.
2. *fpu*: es la unidad de coma flotante.
3. *ccx*: también conocido como crossbar, es el encargado de manejar la comunicación entre las unidades funcionales.

La siguiente línea del script, es la que sintetizará *OpenSPARC*.

```
1 rxil -device=XC5VLX110T sparc fpu ccx
```

Por defecto *OpenSPARC* ya viene sintetizado con otro compilador, el resultado de esta síntesis anterior es el fichero *sparc.edf*. Pero se quiere utilizar la síntesis generada por nosotros. Para que en el proceso de generación del *bitstream* reconozca la síntesis generada con el compilador de *Xilinx*, es necesario modificar dos ficheros.

Las siguiente líneas de código explican cómo utilizar la nueva síntesis. En la segunda línea se copia la síntesis previamente generada al directorio del dispositivo que se está utilizando, *XC5VLX110T*. En las dos últimas líneas se modifica el fichero *iop_fpga_v2_1_0* para que contenga la referencia a la nueva síntesis en lugar de la que ya estaba generada por defecto.

```
1 cd ${DV_ROOT}/design/sys/edk/pcores/iop_fpga_v1_00_a
2 mv netlist/sparc.edf .
3 cp ${DV_ROOT}/design/sys/iop/sparc/xst/XC5VLX110T/sparc.
     ngc netlist
4 cd ${DV_ROOT}/design/sys/edk/pcores/iop_fpga_v1_00_a/data/
```

```
5 mv iop_fpga_v2_1_0.bbd tmp.bbd
6 cat tmp.bbd | sed 's/sparc.edf/sparc.ngc/g' >
     iop_fpga_v2_1_0.bbd
```

A continuación se procederá a generar el *bitstream*. Para poder realizar este paso se utilizará *xps*, que es el *IDE* de *Xilinx*. La herramienta *xps* permite la posibilidad de ejecutarse en modo "línea de comandos". Para ello es necesario pasarle la opción *-nw* y a continuación dónde se encuentra el fichero principal del proyecto, *system.xmp*, que viene generado con el propio proyecto de *OpenSPARC*. Esta opción es requerida para que el *script* de compilación funcione de manera automática sin la intervención del usuario. También es necesario añadir la opción *-scr* para que cargue el script *compilar.tcl*

```
1 xps -nw ${DV_ROOT}/design/sys/edk/system.xmp -scr ${
     SCRIPTS}/compilar.tcl
```

El *script compilar.tcl* contiene las órdenes para generar el *bitstream*, línea 1, y generar las librerías necesarias, línea 2; bajo el entorno de compilación *xps*.

```
1 run bits
2 run libs
```

Una vez haya terminado este proceso se dispondrá del *bitstream* válido para la *Virtex-5 ML505*.

5.1.2. Conexión y configuración de la FPGA

En el apartado anterior se detallaba cómo generar el *bitstream* de *OpenSPARC*, que es un requisito necesario para poder continuar.

Para comprobar que el *JTAG* funciona correctamente, debería encenderse una luz de color verde en el conversor de *JTAG* a *USB*. Si ésta se encuentra de cualquier otro color o apagada, el envío de datos no funcionará correctamente, por lo que haría falta revisar la configuración anteriormente explicada[1].

[1]Esto sólo sirve para el conversor que se ha utilizado

Mediante la orden *impact*, se conectará la placa con el *host* y se establecerán los parámetros de configuración y se asignará el *bitstream* generado anteriormente. Todos estos datos se enviarán mediante el *JTAG*.

Debido a que se utiliza un *script* y se desea que todo sea automatizado, los parámetros necesarios para la conexión se han introducido en un fichero llamado *download.cmd*. De esta manera se ejecuta *impact* junto con el parámetro *-batch script* que permitirá ejecutar las órdenes que contenga este fichero.

```
1 impact -batch ${SCRIPTS}/download.cmd
```

El contenido de *download.cmd* es el que se puede observar a continuación. Las tres primeras líneas son las encargadas de establecer la comunicación mediante el *host* y la placa. A continuación se envía el *bitstream* y en la línea 5 se carga el programa enviado. Una vez terminada la configuración se sale del modo *batch* para que el *script* principal pueda proseguir.

```
1 setMode -bscan
2 setCable -p auto
3 identify
4 assignfile -p 5 -file ${DV_ROOT}/design/sys/edk/
     implementation/download.bit
5 program -p 5
6 quit
```

Una vez terminado el proceso, se habrá cargado el *bitstream* en la placa y se podrá proceder a enviar un programa específico (mirar sección 5.1.3), o cargar un sistema operativo (consultar sección 5.2).

5.1.3. Ejecución y captura de resultados de un programa standalone

Una vez se tiene la placa arrancada y configurada, mirar sección 5.1.2, se puede cargar cualquier programa diseñado para trabajar sobre *OpenSPARC*. A continuación se explicará cómo enviar un programa específico, en este caso uno de los más simples y conocidos, el "hola mundo".

A continuación se puede ver como en la línea de comandos mediante la orden *xmd*, se le pasan como parámetros la información del proyecto y a continuación una serie de órdenes que están escritas en un *script*, *xmd_microblaze_0*.

```
1 xmd -xmp ${DV_ROOT}/design/sys/edk/system.xmp -opt ${
    SCRIPTS}/xmd_microblaze_0.opt
```

Seguidamente se detalla el contenido del *script xmd_microblaze_0*. Este script conecta el *host* con la *FPGA*, tal y como se ve en la línea 1, dónde se indica mediante *mb* que se conecte al *MicroBlaze*, mientras que *mdm* indica el tipo de conexión. Seguidamente se descargan los datos necesarios para que funcione, primero se descarga el firmware, luego el programa que se va arrancar, en este caso el "hola mundo", y finalmente el código del *hypervisor*. Notar que el código del programa específico y del *hypervisor* vienen con una dirección de memoria, que es donde se deben cargar. El tema de las direcciones de memoria viene perfectamente detallado en el documento *DV_GUIDE* [12] en el apartado 6.4 .

```
1 connect mb mdm
2 dow /home/tmp/design/sys/edk/ccx-firmware/executable.elf
3 dow -data /home/tmp/design/sys/edk/examples/bin/
    hello_world.mem.image.gz 0x8af00000
4 dow -data /home/tmp/design/sys/edk/os/proms/1c1t_prom.bin
    0x8ff00000
5 run
6 exit
```

En la penúltima línea se puede ver cómo se arranca la *FPGA*, y el programa empieza a correr. Finalmente se sale del *script xmd_microblaze_0*, para que el *script* principal, *compila.sh*, se pueda dedicar a obtener los resultados.

La obtención de resultados se hace capturando la salida del puerto serie mediante el programa *minicom*. En el siguiente fragmento, se puede ver cómo se debe ejecutar *minicom* capturando la salida en el fichero definido en *${F_MINICOM}*, el cual será leído a posteriori por el *script* pasado un cierto tiempo. El *minicom* debe ser lanzado de forma manual y dejarlo en *background*.

```
minicom -C ${F_MINICOM} &> /dev/null &
```

Para el programa *standalone* elegido se considera que un minuto es más que suficiente para probar que funciona. Pasado este minuto, mediante *xmd*, se ejecuta otro *script* que permite la parada de la *FPGA* para poder continuar con el transcurso del *script* principal. El tiempo de parada viene definido mediante la variable *$TIME_WAIT*.

```
sleep ${TIME_WAIT}
xmd -opt ${SCRIPTS}/xmd_stop.opt
```

El contenido del *script xmd_stop.opt*, son dos sencillas instrucciones, *stop* para parar la *FPGA* y *exit* para salir del modo *script*.

```
stop
exit
```

La figura 5.1 muestra como en el fichero definido como *${F_MINICOM}*, contiene el resultado de la ejecución.

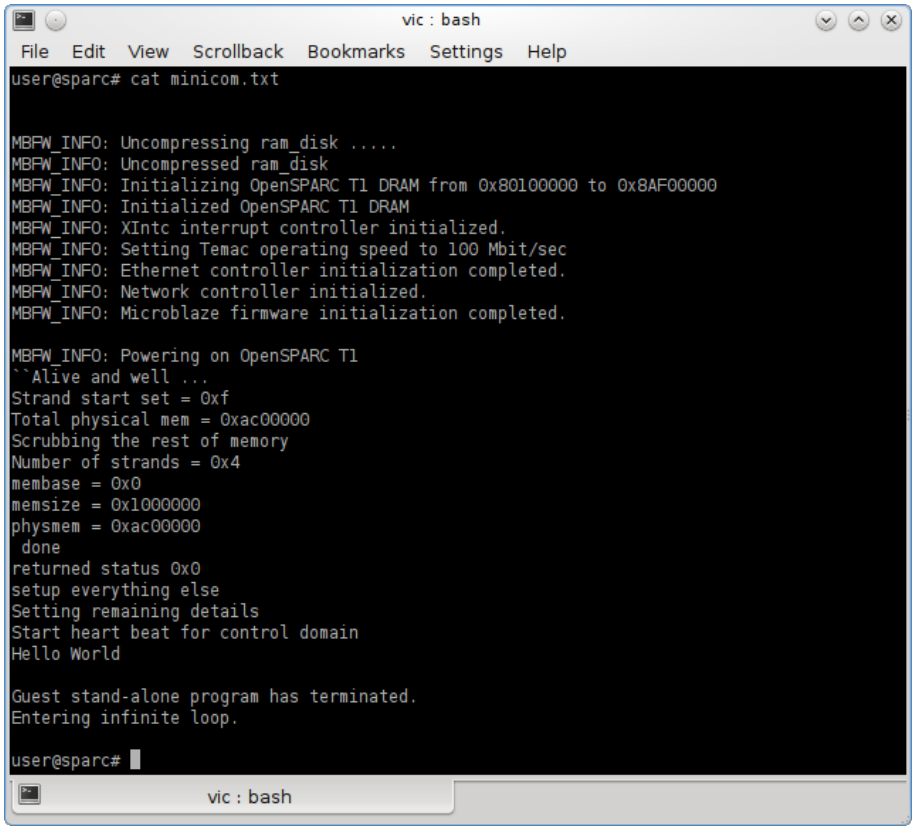

Figura 5.1: *Salida del fichero F_MINICOM*

5.2. Arranque de un Sistema Operativo - Ubuntu

Junto al código fuente de *OpenSPARC* viene una versión adaptada y funcional de *GNU/Linux* y otra sobre *OpenSolaris*. La distribución de *GNU/Linux* utilizada es una *Ubuntu 7.10* recortada para que funcione sobre la *Virtex-5*, ya que ésta no es tan potente como un ordenador.

Es totalmente necesario para poder continuar que la *FPGA* esté configurada y arrancada, mirar apartado 5.1.2.

Una vez conectado, se ejecuta la instrucción *xmd*, que permite el envío de los datos a la *FPGA*. Se le envía como parámetro el proyecto, ya detallado en el apartado anterior y que se muestra a continuación.

```
1 xmd -xmp system.xmp
```

El conjunto de instrucciones que se mostrará a continuación sirven para enviar los datos a la placa. Se envía el *firmware executable*, línea 2, el binario que contiene la información de 1 *core* y 1 *thread* para el *bitstream* que se ha generado y enviado con anterioridad, y finalmente en la penúltima línea la imagen comprimida del *Sistema Operativo*.

Destacar que en el caso de 1 *core* 1 *thread* y de la imagen del *Sistema Operativo*, se añade la dirección de memoria donde debería ubicarse.

La única diferencia con la carga de un programa *standalone* (mirar el apartado 5.1.3), es que en lugar de enviar los datos de nuestro programa, se envía el Sistema Operativo, ramdisk.ubuntu-7.10-gutsy.gz. Esto se observa en la línea 4. Finalmente con *run* se arranca la placa y empieza a arrancar el sistema operativo.

```
1 connect mb mdm
2 dow ccx-firmware/executable.elf
3 dow -data os/proms/1c1t_obp_prom.bin 0x8ff00000
4 dow -data os/Ubuntu/7.10-Gutsy/proto/ramdisk.ubuntu-7.10-
     gutsy.gz 0x8af00000
5 run
```

En este proceso de carga de un Sistema Operativo, al no ejecutarse en ningún *script*, no es necesario salir de la consola y volver al *prompt*. Además si se pusiera un *time out*, el Sistema Operativo se apagaría de golpe y no permitiría interactuar con él.

En caso de querer arrancar *Open Solaris* en lugar de *Ubuntu*, el proceso es igual al explicado anteriormente para la carga de *Ubuntu*, sólo que substituyendo la tercera línea por la siguiente.

```
1 dow -data os/OpenSolaris/proto/ramdisk.snv-b77-nd.gz 0
     x8af00000
```

En la figura 5.2, se puede ver cómo comienza el proceso de arranque. Una vez haya terminado, es necesario decirle que arranque el Sistema Operativo, un proceso manual en el que hay que escribir "ok boot" para ambos Sistemas Operativos, *Ubuntu* y *OpenSolaris*. Lo que implica que debe estar conectado el *minicom*, o *cutecom* si se desea trabajar desde un sistema de ventanas.

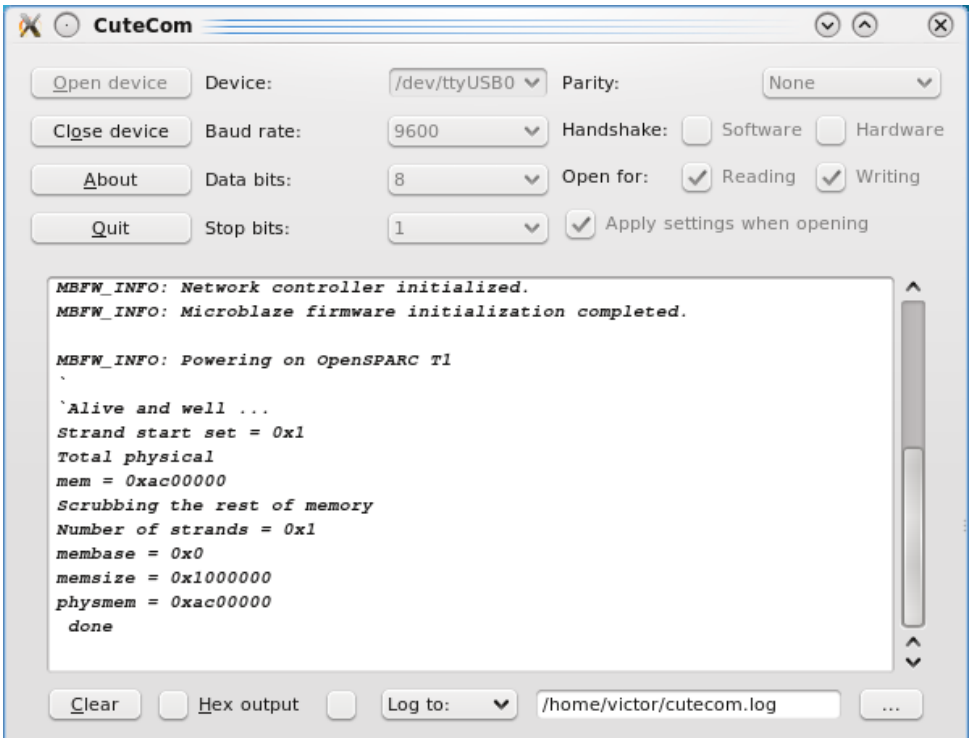

Figura 5.2: *Una vez ha cargado el OpenSPARC en la FPGA*

Una vez comienza la carga del sistema operativo, se demora un rato. Y aún habiendo ofrecido el *prompt* de *login* sigue cargando servicios, como puede ser *ssh*, que es uno de los últimos.

La figura 5.3 ofrece el *prompt* de login tras el largo proceso de carga del Sistema Operativo. Por defecto el usuario de acceso es *root* con *password root*.

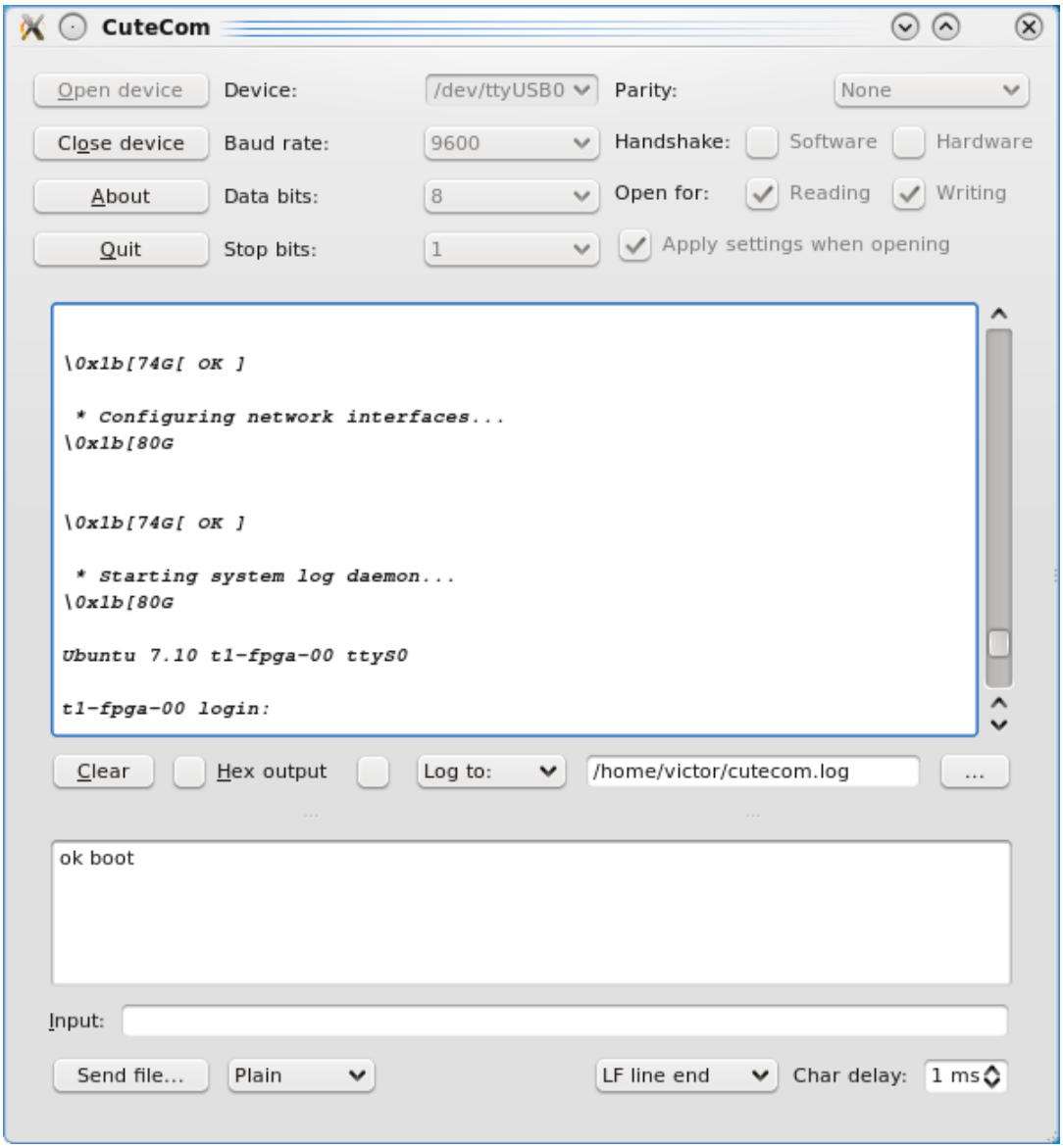

Figura 5.3: *FPGA arrancada con el Sistema Operativo*

Una vez finalizado se puede interactuar con el Sistema Operativo, como si el propio se tratase, con la diferencia de que es un poco más lento a la hora de la ejecución y de no disponer de tantos comandos.

Finalmente, cuando se quiera parar, se vuelve a la consola y en la línea de comandos que se había dejado en *stand by*, se le envía la orden de *stop* y finalmente *exit* para salir.

```
stop
exit
```

Todos los cambios hechos durante la carga del Sistema Operativo se perderán a no ser que se guarden en un soporte físico. Por defecto todo se carga en la memoria *RAM*.

En el apartado 5.2.2 se explica cómo añadir/modificar aplicaciones o configuraciones personalizadas del Sistema Operativo *Ubuntu*.

5.2.1. Código específico

En este apartado, se muestra como compilar un código, recibirlo en la placa y ver que funciona correctamente. A diferencia del arranque de un programa *standalone* éste se ejecuta desde la línea de comandos del Sistema Operativo que está corriendo *OpenSPARC*.

Un requisito indispensable para poder continuar es haber arrancado la *FPGA* con el sistema operativo tal y como se ha descrito anteriormente.

El siguiente código muestra por pantalla "hola mundo".

```
#include <stdio.h>

int main (void) {
        printf("Hola mundo \n");
        return 1;
}
```

Se compilará usando el *cross-compiler* que se ha instalado (mirar el apartado 5.3).

```
sparc-linux-gcc -o holaMundo holaMundo.c
```

La figura 5.4 muestra cómo desde la *FPGA*, accediendo por *ssh*, se puede recibir un código mediante *FTP*.

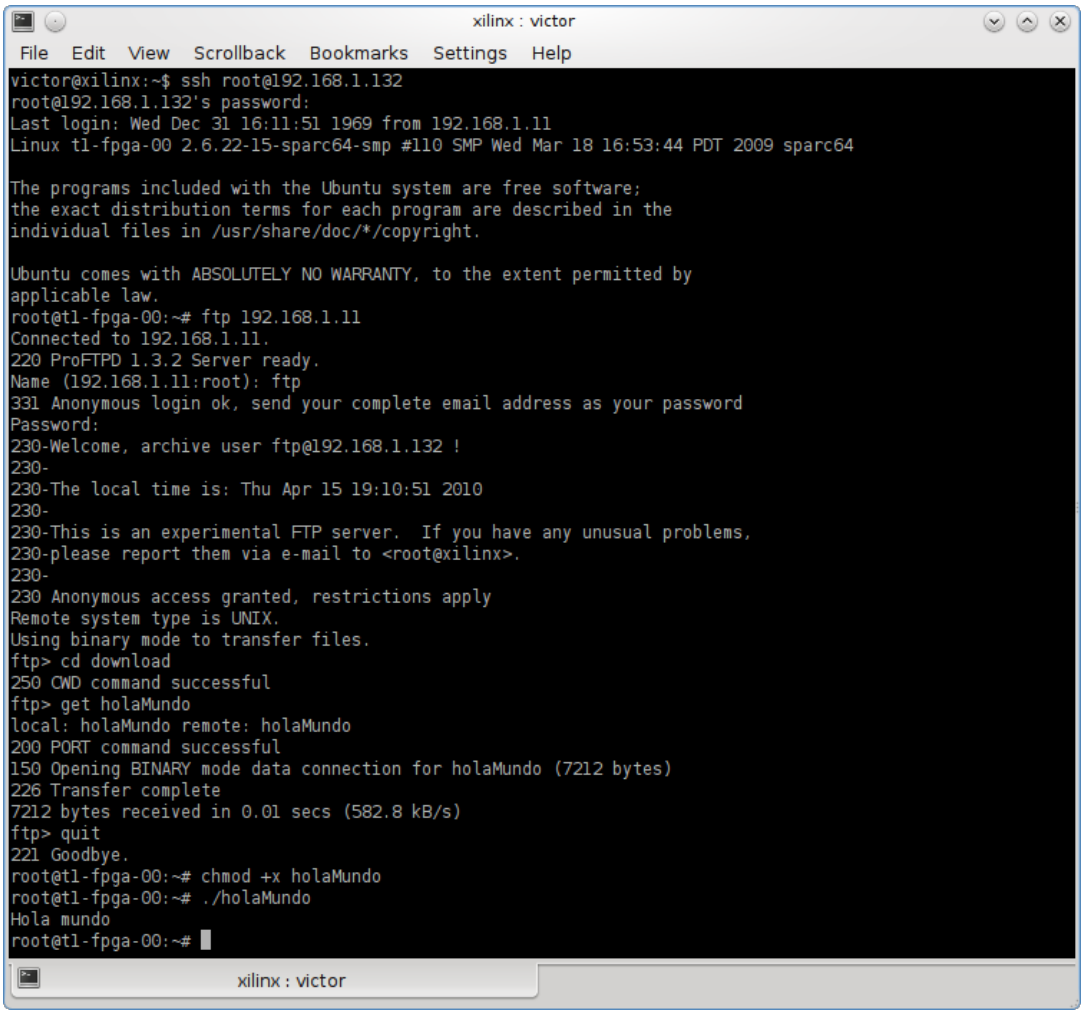

Figura 5.4: *Ejecutando un programa de prueba en el Sistema Operativo de la FPGA*

5.2.2. Modificación del Sistema Operativo

Cada vez que se arranca la imagen de *Ubuntu* en la *FPGA* no se guardarán los cambios hechos durante su ejecución, a excepción de que use algún dispositivo externo de almacenamiento. Sin embargo, puede ser que interese hacer ciertas modificaciones sobre el Sistema Operativo, para no tenerlas que hacer cada vez que se

arranca la placa.

Para poder modificar la imagen de *Ubuntu* que viene por defecto, tan sólo es necesario descomprimirla sobre un directorio. En este directorio se puede observar que está la jerarquía de directorios propia de *Ubuntu*. De tal manera que sólo es necesario hacer los cambios necesarios sobre esta estructura, es decir, agregar, eliminar, modificar archivos y directorios. Finalmente se vuelve a comprimir la imagen y ya estaría todo listo.

La parte más complicada del proceso de la modificación es la instalación de algún paquete. Esto se debe a que esta distribución se encuentra recortada y no se dispone de un gestor de paquetes, lo que hace obligatorio instalar manualmente todo aquello que se desee. La no existencia de un gestor de paquetes implica tener que comprobar manualmente todas las dependencias, lo que es un proceso bastante pesado. Este proceso puede recordar a distribuciones como *slackware* hace algunos años, cuando este proceso era manual.

Para hacer la instalación, es necesario conocer el sistema de dependencias de GNU/Linux. Cada paquete que se quiera instalar, puede depender de otros y puede ser que estos no se encuentren instalados y a su vez estos dependan de otros. Por lo que hará falta ir instalándolos uno a uno. Este proceso recursivo termina en el momento que esté todo lo necesario instalado. Notar que algunas dependencias son opcionales y se pueden evitar, por lo que se ahorra algo de espacio y dolor de cabeza a la hora de tener que instalar demasiadas cosas.

¿Cómo saber de lo que depende el paquete/programa que se va a instalar? Las siguientes líneas descomprimen el paquete en la carpeta *tmp* y a continuación listan todas las dependencias de los binarios que éste incluye.

```
1 mkdir tmp
2 dpkg --extract archivo.deb tmp/
3 ldd ‘find ./tmp/ -type f | grep bin ‘
```

¿Cómo saber si cumple las dependencias? Mediante *find* se pueden buscar si los archivos de los que dependen nuestros binarios se encuentran o no instalados. Es un proceso bastante pesado, pero puesto que la distribución de la que hablamos está muy reducida, no existe otra opción.

En el caso de querer instalar *gdb*, haría falta buscar el correspondiente paquete en la web http://packages.ubuntu.com/, descargarlo y extraerlo en la carpeta donde se ha descomprimido *Ubuntu*.

En el siguiente fragmento se detallan los pasos para añadir *gdb* a nuestra imagen. Primero se accede al directorio donde se encuentra la imagen, de las líneas dos a cuatro, se descomprime y se monta en el directorio *tmp*. Acto seguido se descomprime el paquete del gdb, también se podría decir que se instala, finalmente se guardan los cambios y se vuelve a comprimir la imagen.

```
1 cd ${DV_ROOT}/design/sys/edk/os/Ubuntu/7.10-Gutsy/proto/
2 gunzip  ramdisk.ubuntu-7.10-gutsy.gz
3 mkdir mnt
4 sudo mount ramdisk.ubuntu-7.10-gutsy ./mnt
5 dpkg --extract gdb_6.4-1ubuntu5.1_sparc.deb ./mnt/
6 sync
7 sudo umont ./tmp
8 gzip ramdisk.ubuntu-7.10-gutsy
```

En este caso es suficiente ya que si se comprueban todas las dependencias de las que depende *gdb*, éstas se cumplen. Un consejo altamente recomendable, es buscar los paquetes para dicha distribución y la arquitectura correspondiente.

5.3. Cross-compiler

Un *cross-compiler* es un compilador como cualquier otro, tan sólo que trae la peculiaridad de que permite compilar programas para otras arquitecturas diferentes a la arquitectura donde se está compilando.

A continuación se detallará cómo crearlo. Nuestra experiencia es que el *cross-compiler* es razonable para programas pequeños. Cuando estos tienen un tamaño considerable y muchas librerías que *linkar*, se puede volver un proceso bastante complicado, ya que hay que preparar todo un sistema bajo la arquitectura *SPARC*.

Antes de poder compilar el compilador es necesario:

1. Los binarios esenciales, como *as*, *ld*, *ar*, entre otros.

2. La *glibc*, que es la librería encargada de gestionar la memoria dinámica.

3. Las fuentes del *kernel* sobre el cual se va a trabajar.

Todo el proceso de generación, descarga y compilación se hará en el directorio "cross". Para ello se creará tal directorio

Lo primero de todo es descargarse el paquete de las *binutils*, que son los binarios básicos del sistema, mediante la orden *wget* y la *url* donde se encuentra.

```
1 wget http://ftp.gnu.org/gnu/binutils/binutils-2.19.tar.bz2
```

A continuación se descomprime el paquete de las *binutils* descargadas. En las dos siguientes lineas se crea la carpeta "build-binutils"y se accede a ella. Aquí dentro es donde se compilarán las herramientas. Se hace en un directorio separado para no mezclar código con ejecutables.

```
1 mkdir build-binutils
2 tar jxf binutils-2.19.tar.bz2
3 cd build-binutils
```

El siguiente paso define las variables *$PREFIX* y *$TARGET*. La primera define el lugar donde se instalará todo y la segunda define el comienzo del nombre que tendrán los ejecutables.

```
1 export PREFIX=/usr/local
2 export TARGET=sparc-linux
```

El siguiente paso define una compilación típica bajo cualquier sistema **NIX*. Primero se configura con las opciones anteriormente definidas, se compila y en la línea 3 se procede a instalarlo si todo ha ido correctamente.

```
../binutils-2.19/configure --target=$TARGET --prefix=
   $PREFIX -v
make
make install
cd ..
```

Una vez instalado, se puede ver en la carpeta */usr/local/* cómo se encuentran los ficheros recientemente instalados.

El siguiente paso consiste en conseguir la librería dinámica y sus cabeceras, líneas 1 y 2, que contienen principalmente el código ensamblador que se entiende con la arquitectura *SPARC*. El resto de instrucciones desempaqueta los paquetes en su correspondiente directorio.

```
wget https://launchpad.net/ubuntu/+source/glibc/2.6.1-1
   ubuntu10/+build/425809/+files/libc6_2.6.1-1
   ubuntu10_sparc.deb
wget https://launchpad.net/ubuntu/+source/glibc/2.6.1-1
   ubuntu10/+build/425809/+files/libc6-dev_2.6.1-1
   ubuntu10_sparc.deb
mkdir libc6_2.6.1-1ubuntu10_sparc
mkdir libc6-dev_2.6.1-1ubuntu10_sparc
dpkg --extract libc6_2.6.1-1ubuntu10_sparc.deb libc6_2
   .6.1-1ubuntu10_sparc
dpkg --extract libc6-dev_2.6.1-1ubuntu10_sparc.deb libc6-
   dev_2.6.1-1ubuntu10_sparc
```

El siguiente paso es copiar las cabeceras al directorio que se utilizará para instalar todo lo necesario para el *cross-compiler*, que viene definido por *$PREFIX* y que las rutas sean las correctas.

Ahora se procede a copiar las cabeceras que se necesitarán, en el directorio dónde se instalará todo, *$PREFIX/sparc-linux/*.

```
cp -a libc6-dev_2.6.1-1ubuntu10_sparc/usr/include/ $PREFIX
   /sparc-linux/
cp -a /usr/src/linux-headers-2.6.31-14/include/linux/
```

```
    $PREFIX/sparc-linux/include/linux
cp -a /usr/src/linux-headers-2.6.31-14/arch/sparc/include/
    asm/ $PREFIX/sparc-linux/include/asm
cp -a /usr/src/linux-headers-2.6.31-14/include/asm-generic
    / $PREFIX/sparc-linux/include/asm-generic/
cp -a libc6_2.6.1-1ubuntu10_sparc/lib/* $PREFIX/sparc-
    linux/lib
cp -a libc6-dev_2.6.1-1ubuntu10_sparc/usr/lib/* $PREFIX/
    sparc-linux/lib/
```

Modificar el fichero *libc.so* para que apunte a las librerías correctas. Dejándolo tal y como se puede ver a continuación.

```
/* GNU ld script
Use the shared library, but some functions are only in
the static library, so try that secondarily.  */
OUTPUT_FORMAT(elf32-sparc)
GROUP ( /usr/local/sparc-linux/lib/libc.so.6 /usr/local/
    sparc-linux/lib/libc_nonshared.a  AS_NEEDED ( /usr/local
    /sparc-linux/lib/ld-linux.so.2 ) )
```

Seguidamente se descargará el código fuente del compilador *gcc* y se extraerá para compilarlo posteriormente.

```
cd ~/cross
wget ftp://ftp.irisa.fr/pub/mirrors/gcc.gnu.org/gcc/
    releases/gcc-4.2.4/gcc-4.2.4.tar.bz2
tar jxf gcc-4.2.4.tar.bz2
```

Finalmente se compila e instala el compilador. Se crea un directorio donde compilarlo, para mantener separado el código fuente de los binarios generados, y en las tres últimas líneas se compila, con las opciones de *c* y *c++*, que son las más utilizadas, a parte de las opciones que se han comentado anteriormente para definir el nombre y la ubicación, *$PREFIX* y *$TARGET*.

```
mkdir build-gcc
cd build-gcc
../gcc-4.2.4/configure --enable-languages=c,c++ --disable-
   libgomp --target=$TARGET --prefix=$PREFIX -v
make all
make install
```

Con esto ya estaría instalado el *cross-compiler*. Para utilizarlo es necesario que la ruta *$PREFIX/bin* se encuentre en la variable *PATH*.

A partir de ahora, para compilar cualquier programa para la arquitectura *SPARC*, se debería compilar con *sparc-linux-gcc*, como se muestra a continuación.

```
sparc-linux-gcc -o miFicheroBinario miFuente.c
```

Una vez generado el ejecutable, se puede comprobar que es valido mediante el comando *file*, que mostrará información referente al ejecutable que se le pase como parámetro.

```
file miFicheroBinario
miFicheroBinario: ELF 32-bit MSB executable, SPARC32PLUS,
   V8+ Required, version 1 (SYSV), dynamically linked (uses
    shared libs), for GNU/Linux 2.6.8, not stripped
```

Si todo ha ido correctamente, se podrá observar como indica que es válido para *SPARC32PLUS*, además de poder ser ejecutado con *qemu* [2]. La figura 5.5 muestra cómo compilar un programa y probar que éste es valido para la arquitectura *SPARC*.

[2]Qemu es un emulador para diferentes arquitecturas

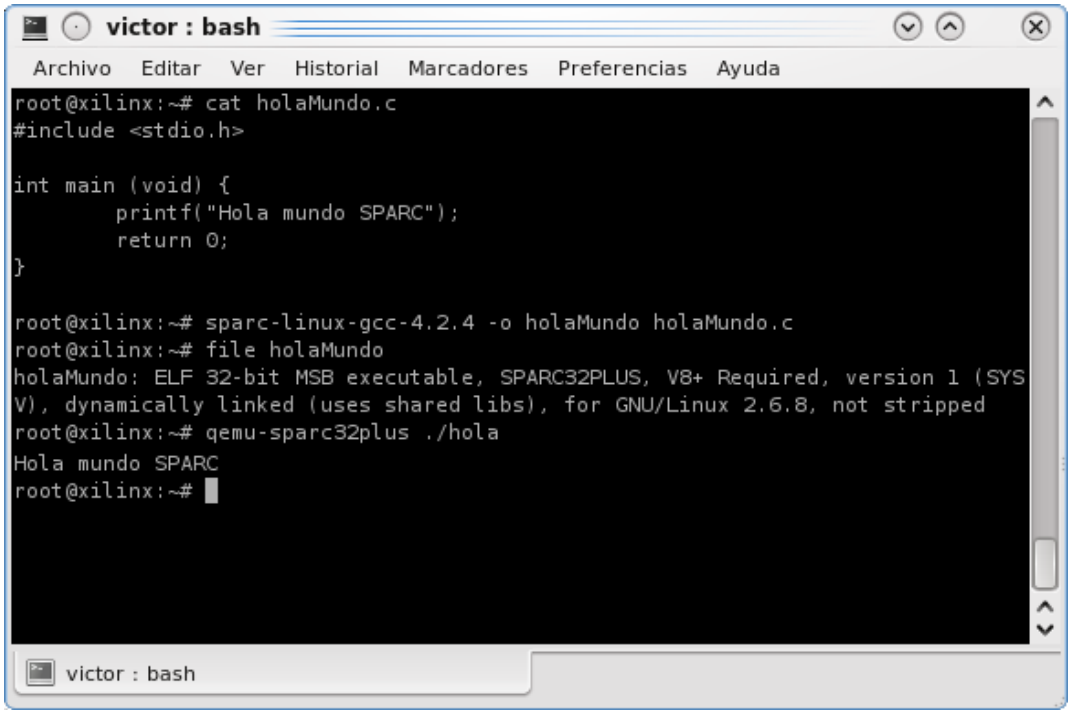

Figura 5.5: *Resultado del Hola Mundo desde consola*

6. Portal Web

Con este portal web se pretende sacar el máximo partido a una placa, de esta manera automatizando y haciendo transparente todos los procesos antes mencionados.

El portal web permite el envío de código, obtener unos resultados, (sobre el proceso de compilación, envío de datos y ejecución), y mantiene un histórico de las compilaciones previas.

6.1. Instalación

En la siguiente sección se explicará cómo instalar el portal, y posteriormente se explicará cómo usarlo adecuadamente, ya sea para un usuario o para gestionar los usuarios que pueden acceder al portal.

Este portal web ha sido desarrollado con *html + css + php + mysql*. El siguiente comando sirve para instalar los paquetes necesarios. Por defecto, en su instalación, ya vienen preconfigurados de una manera que es perfectamente válida.

```
1 sudo aptitude install apache2 php5 php5-mysql mysql-server
    mysql-client
```

El siguiente paso consiste en copiar todos los archivos del funcionamiento de la web en su correspondiente directorio, */var/www*, y dar los permisos adecuados a la estructura. Esto se puede ver aquí:

```
1 cp -rf $archivos_web/* /var/www
2 chmod 775 /var/www/outputs/
3 chmod 775 /var/www/archivos/
```

Los directorios *archivos* y *outputs* son los que utiliza el portal web para guardar los códigos recibidos de los usuarios y los resultados. Por lo que el usuario necesitará permiso de lectura y escritura.

El usuario encargado de hacer todo el proceso interno de la web es *sparc*. Este usuario debe pertenecer a varios grupos que le permitirá tener acceso al puerto *RS-232*, acceso a los ficheros de la web para recoger nuevos códigos y guardar los resultados. A continuación se detallan los comandos utilizados:

```
1 sudo adduser sparc
2 sudo gpassw -a sparc scanner,users,lp,www-data
3 sudo passwd sparc
```

En la *$HOME* del usuario *sparc* se guardarán los *scripts* de compilación, para que el proceso sea automático. Estos *scripts* son *compila.sh* y *lanzar.sh*, tal y como se puede ver en el siguiente fragmento. El primero ha sido debidamente explicado en el apartado de compilación 5.1, mientras que el segundo se dedica a recoger los códigos del usuario y enviar los resultados, dejando un *log* al administrador para que pueda ir siguiendo el proceso más detalladamente.

```
1 cp compila.sh lanzar.sh $HOME_SPARC
```

A continuación se detallará el contenido del *script lanzar.sh*.

En el siguiente fragmento se puede ver cómo se crea un fichero llamado *running.txt* que informa al administrador de cuándo se empezó a ejecutar el *script* y previene que se ejecute de nuevo el *script* cuando éste aún está en proceso. Al final del *script* este fichero será borrado. Sólo es un indicador.

```
1  #!/bin/sh
2 SENAL="/var/www/running.txt"
3 if [ -f ${SENAL} ]; then
4         echo "Hay otro proceso corriendo"
5         exit 1
6 fi
7
8 FECHA=`date +%d/%m/%Y-%H:%M`
```

```
9  echo "running since ${FECHA}" >> ${SENAL}
```

Seguidamente se definen variables para que en caso de querer modificar algún parámetro sea un proceso más sencillo. El uso de las variables viene explicado en el fragmento que viene a continuación.

```
1  ## Lugar donde estan los scripts
2  INIT="/home/victor/sparc"
3  ## Donde se guardan las salidas de cada ejecucion
4  OUTPUT="${INIT}/outputs"
5  ## Directorio del OpenSPARC precompilado
6  PRECOMPILED="${INIT}/precompiled"
7  ## Directorio temporal donde se compila
8  TEMPORAL="/home/tmp"
9  ## Directorio de donde se leen los archivos
10 ARCHIVOS="/var/www/archivos"
11 ## Directorio donde se guardan los resultados de cada
      ejecucion que leera la pagina web.
12 OUTPUTS_WWW="/var/www/output"
13 ## Fichero donde se guarda todo registro de lo que sucede
14 OUTPUT_LOG="/var/www/logCompilation.txt"
15 # Tiempo de espera para la ejecucion de un programa
16 TIME_WAIT="1m"
17
18 FECHA=‘date +%d/%m/%Y-%H:%M‘
```

A continuación se escribe en el *log* cuando comienza a ejecutarse este *script*.

```
1  echo "${FECHA} START compilation/execution of waiting
      works" >> ${OUTPUT_LOG}
```

En el siguiente paso se almacenan el nombre de todos los ficheros enviados mediante el formulario de la web en la variable *$FILES* para que un proceso iterativo vaya ejecutándolos y guardando sus resultados.

```
1  FILES=‘ls ${ARCHIVOS}/*.tar.bz2‘
2  for i in ${FILES}
3  do
```

A partir de aquí es donde comienza el proceso que irá repitiendo por cada compilación. Lo primero de todo es eliminar cualquier residuo de una anterior ejecución para que el resultado sea valido.

```
cd ${TEMPORAL}
## Elimina residuos de anteriores compilaciones
rm -rf ${TEMPORAL}/*
```

Seguidamente se copia el fichero preparado de *OpenSPARC*.

```
## Copia el precompilado al directorio temporal
echo "Copiando precompilado..."
cp -rf ${PRECOMPILED}/* ${TEMPORAL}

## DESCOMPRIMIR_PRECOMPILED=‘ls *.tar.bz2‘
tar -jxf ${DESCOMPRIMIR_PRECOMPILED}
rm -f ${DESCOMPRIMIR_PRECOMPILED}

## Copia el source al directorio temporal y como registro
   donde los outputs
echo "Copiando fichero...${i} a ${TEMPORAL}"
cp ${i} ${TEMPORAL}
```

El siguiente fragmento de código limpia el nombre del archivo original, quitándole la extensión para usarlo posteriormente a la hora de generar los ficheros de salida.

```
NOMBRE=‘ls *.tar.bz2‘
echo $NOMBRE
re_match=".tar.bz2"
replace=""
TXT_FORMATEADO=$(echo ${NOMBRE} | sed -e "s/$re_match/
   $replace/");
```

A continuación se guarda el fichero original para mantener un histórico de los códigos del usuario.

```
## Copio el original para mantener un historico del codigo
    enviado
## Se guarda en $outputs_www
cp "${TEMPORAL}/${NOMBRE}" "${OUTPUTS_WWW}/${
   TXT_FORMATEADO}_source.tar.bz2"
```

A partir de ahora se comienza a trabajar con el código del usuario. Primero de todo se descomprime el código del usuario y acto seguido se lanza el *script* de compilar, explicado previamente en el apartado 5.1

```
## Empieza a trabajar con el fichero
## Descomprime, y compila (este tambien lo envia a la
   placa)
echo "Descomprimiendo... ${NOMBRE}"
tar -jxf ${NOMBRE}
cd ${INIT}
echo "lanzando: ${NOMBRE} -"
bash ./compila.sh
```

En el siguiente conjunto de órdenes se puede ver como se guardan los resultados en la web y en el *log* de compilaciones. En la línea 5, se puede ver cómo comprime todos los ficheros que pueden proporcionar alguna salida y se guardan en la carpeta de destino, que es la carpeta definida como *outputs*. Finalmente el archivo *result.txt* contiene el resultado general de cada proceso que ha sido lanzado, cómo compilar, ejecutar, etc. y lo guarda en el apartado de *outputs* para que más tarde la web pueda leerlo.

```
## Guarda los resultados
cd ${OUTPUT}
## Guarda en un comprimido los siguientes archivos -
   minicom.txt jtag.txt compilacion.txt rxil.txt
DESTINO="${OUTPUTS_WWW}/${NOMBRE}"
tar -cf ${DESTINO} minicom.txt jtag.txt compilacion.txt
   rxil.txt
## Anade al registro la salida de la compilacion actual
echo "Trabajando con el fichero... ${i}" >> ${OUTPUT_LOG}
cat result.txt >> ${OUTPUT_LOG}
```

```
## Copia el registro de la compilacion para que la web lo parsee
cp result.txt ${OUTPUTS_WWW}/${TXT_FORMATEADO}_result.txt
```

Una vez ha terminado con el código del usuario se procede con el siguiente código, hasta que finalmente no queda ningún otro código. Entonces guarda la fecha en el registro principal y borra el indicador (running.txt) que evita que este *script* sea lanzado durante su ejecución.

```
done
FECHA=`date +%d/%m/%Y-%H:%M`
echo "${FECHA} STOP compilation/execution of waiting works " >> ${OUTPUT_LOG}
rm -f ${SENAL}
```

A continuación se crea la base de datos y toda su estructura para que el portal web pueda funcionar, ya que depende de la base de datos para que funcione la validación, reconozca el tipo de usuario que se ha validado, hasta incluso saber qué idioma prefiere el usuario.

Primero de todo se accederá a la consola de *mysql*, mediante la siguiente instrucción:

```
mysql -u root -p
```

Una vez con acceso a la base de datos, crearemos la base de datos que funcionará bajo el portal web, esta base de datos se llama "opensparc".

```
CREATE DATABASE `opensparc` DEFAULT CHARACTER SET latin1 COLLATE latin1_spanish_ci;
USE `opensparc`;
```

Una vez se ha creado la base de datos, se procede a crear la tabla.

```
CREATE TABLE IF NOT EXISTS `users` (
  `user` varchar(30) NOT NULL,
```

```
3     `pass` varchar(50) NOT NULL,
4     `group` varchar(30) NOT NULL,
5     `language` varchar(30) NOT NULL,
6     PRIMARY KEY (`user`)
7   ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

A continuación se crea un usuario administrador para poder gestionar el portal web desde la página de administración. Por defecto, la contraseña está encriptada en *md5*, la que se puede ver a continuación corresponde con los valores "12345".

```
1   INSERT INTO `users` (`user`, `pass`, `group`, `language`)
       VALUES
2   ('administrador', '827ccb0eea8a706c4c34a16891f84e7b', '
       admin', 'sp');
```

Finalmente se crea el usuario que permitirá al portal web acceder a la base de datos, para poder consultar, actualizar, eliminar.

```
1   GRANT SELECT, INSERT, DELETE, UPDATE ON opensparc.* TO '
       pfc'@'localhost' IDENTIFIED BY 'password';
```

Una vez terminado ya estaría todo listo para poder utilizar correctamente el portal web.

6.2. Uso del portal

La página web ha sido diseñada mediante un sistema de validación para que cada usuario disponga de sus propios resultados sin verse afectados por los de los compañeros, y uno o varios usuarios administradores a los que se les permite ver la totalidad de los resultados. La figura 6.1 muestra la pantalla de validación.

La página web ha sido diseñada de manera que existe un banner superior, un menú lateral a la izquierda y a su derecha, el contenido de cada apartado. Consta de dos apartados principalmente, la opción de envío y de resultados. Como su nombre indica, la opción envío sirve para enviar un nuevo código para que éste sea probado y, la opción resultados muestra un histórico de los resultados que se han ido obteniendo.

Figura 6.1: *Página de validación*

Para enviar un nuevo código mediante la web hay que ir a la opción de envío donde aparecerá un formulario que permite enviar un archivo. En la figura 6.2 se puede ver cómo enviar un código. El código debe estar comprimido como *.tar.bz2* para que más tarde los *scripts* sepan como tratarlo.

Cada fichero que es enviado va a parar a la carpeta */var/www/archivos*, en esta carpeta se guardan todos aquellos archivos que hayan sido enviados y aún no hayan podido ser compilados. Cada vez que se envía un fichero, éste es renombrado para que ningún otro pueda sobreescribirlo. El patrón del nombre del fichero es *AAAAMMDD-HHMMSS-$usuario*. A medida que el *script* va ejecutándolos van desapareciendo de esta carpeta para ir a parar a otra donde quedarán para poder ser revisados en momentos posteriores, por defecto esta carpeta es */var/www/outputs*, personalizable según el administrador.

Una vez que se ha enviado el código, el sistema interno de scripts se encarga de hacer lo necesario con ellos. Estos se encargan de hacer todo lo necesario para compilarlo, ejecutarlo, obtener los resultados y finalmente, los dejan en la carpeta *outputs*.

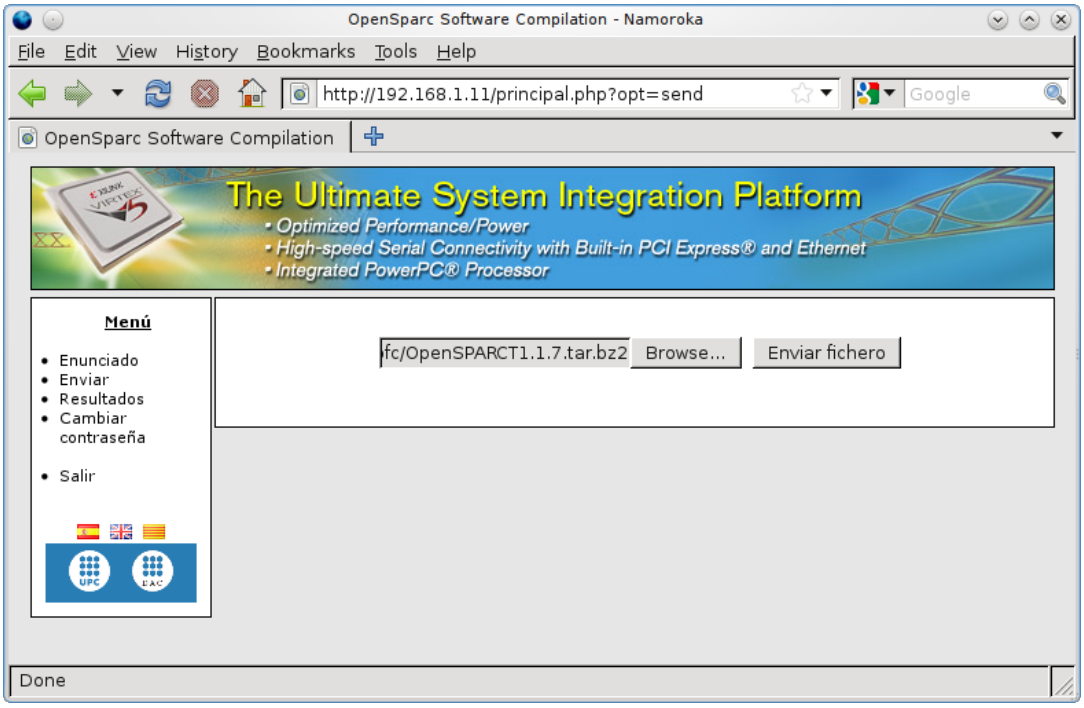

Figura 6.2: *Envío de archivos*

El sistema de *scripts* se basa principalmente en *lanzar.sh* y *compila.sh*. El primero de todos, *lanzar.sh* (ver apartado 6.1 para más detalle), se encarga de borrar el contenido de la carpeta temporal donde se compila cada ejecución, copia un precompilado a dicha carpeta y procede a copiar el primer fichero que se encuentra en */var/www/archivos*, donde lo descomprimirá y ejecutará el segundo *script*, *compila.sh* (ver apartado 5.1 para más detalle). Una vez el *script* haya terminado, recoge los resultados los comprime y los envía a */var/www/outputs/*. Seguidamente sigue con el resto de códigos hasta que no queden más.

La figura 6.3, muestra el resultado de dos compilaciones y su correspondiente histórico. Además, permite descargar, por parte del usuario, los *logs* de las compilaciones para saber exactamente qué ha pasado.

Como se puede ver en la figura 6.3, de cada ejecución se extraen los siguientes resultados por orden de aparición:

1. Fecha: fecha de envío del código
2. Hora: hora de envío del código
3. Usuario: esta columna sólo es visible desde la vista de administrador.

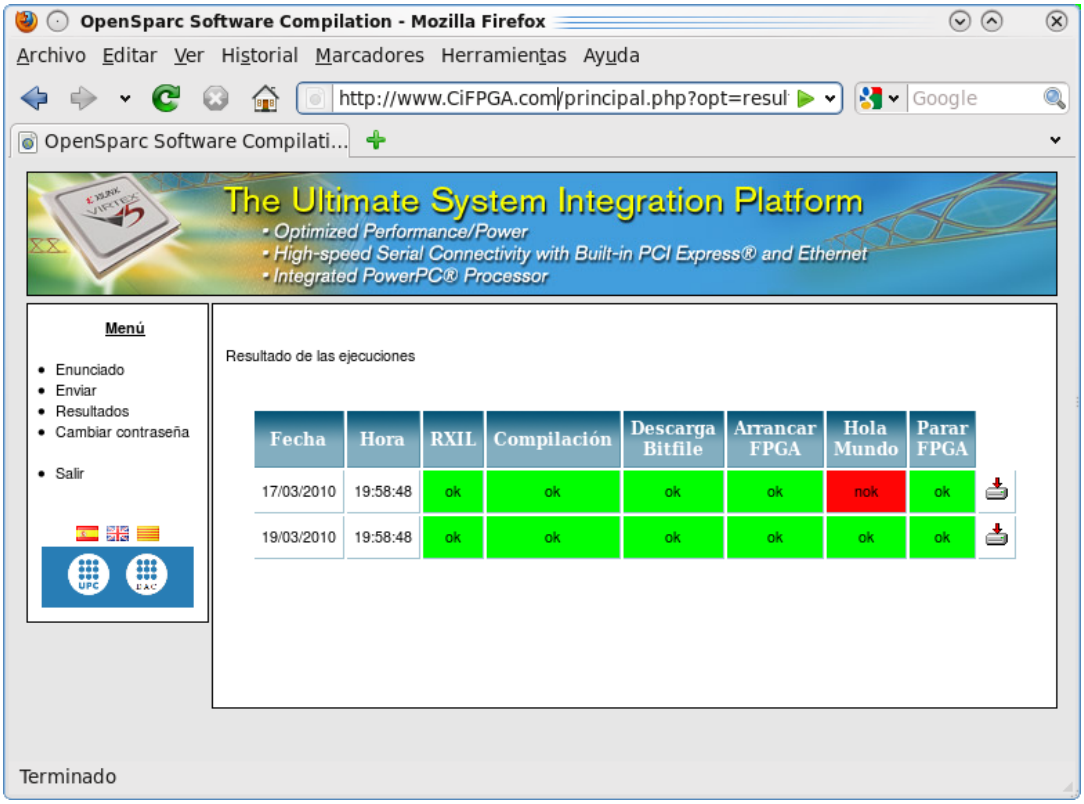

Figura 6.3: *Página de resultados vista por un usuario*

4. *RXIL*: resultado de la síntesis del *netlist*.

5. Generar *bitstream*: resultado de la generación del *bitstream* de *OpenSPARC*.

6. Descargar *bitstream*: se envía el bitstream generado previamente y se obtienen los resultados.

7. Arrancar *FPGA*: una vez enviados todos los datos se arranca la placa y se mira si hay conexión.

8. Ejecutar Hola Mundo: resultado de ejecutar el programa *standalone*.

9. Parar *FPGA*: una vez parada la *FPGA* se guarda el resultado.

Desde el punto de vista del administrador, la opción de resultados contendría los resultados de todos los usuarios que hayan enviado códigos, tal y como se ve en la figura 6.4.

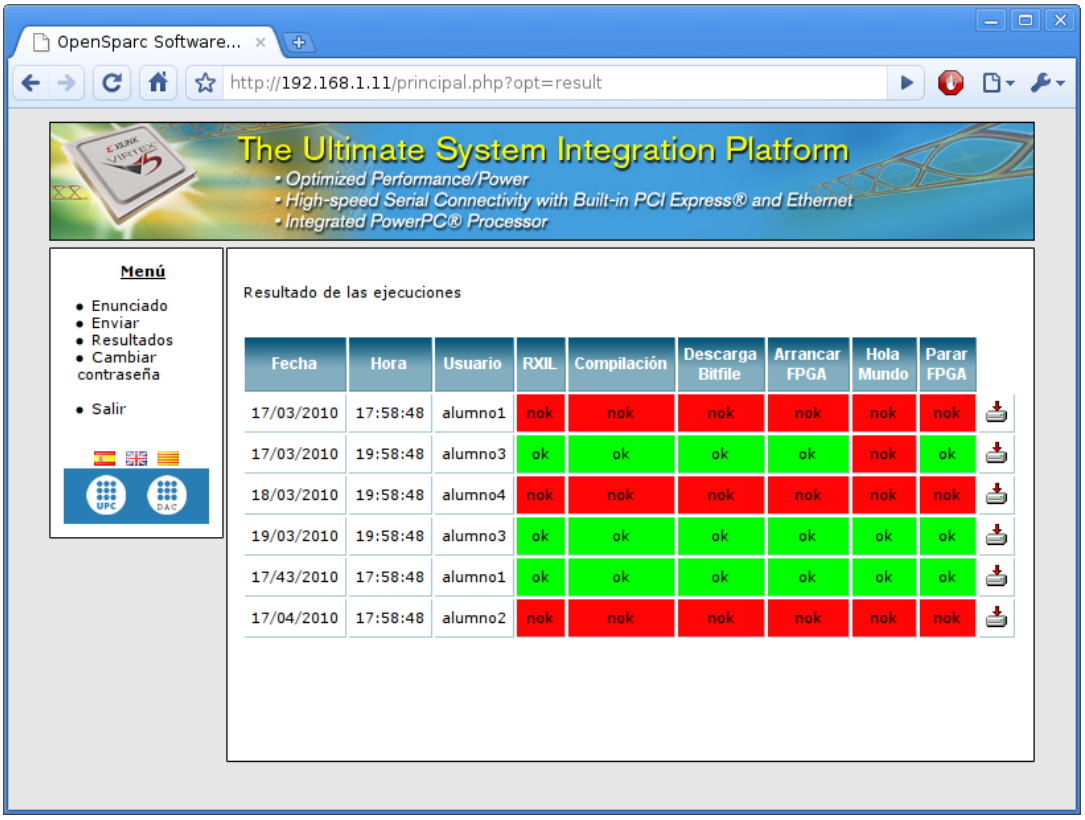

Figura 6.4: *Vista de resultados desde la perspectiva del administrador*

La estructura interna de carpetas y ficheros más importantes relacionados con el portal web son:

- index.php: es la página donde cada usuario se validará para acceder a su propia página de resultados.
- archivos/: aquí es donde van a parar todos los archivos que cada usuario envía para que sea compilado.
- imagenes/: lugar donde se almacenan las imágenes que se muestran en la web.
- includes/: esta carpeta tiene como objetivo guardar ficheros de configuración, como el de acceso a la base de datos, y los ficheros que compondrían cada opción del menú.
- output/: lugar donde se guardan los resultados de cada ejecución, para que la opción de resultados sea capaz de mostrarlos vía web al usuario y permitir su descarga.
- principal.php: contiene el documento inicial de la web después de haberse validado, con la estructura básica.

- style.css: fichero *css* que contiene el diseño de la página.

6.3. Administración del portal

La gestión de usuarios está centralizada en otro apartado de la web, que es la administración. Para acceder a la administración se debe tener un usuario previamente administrador y acceder a la *url* del portal web, añadiendo */adm/* al final.

En la figura 6.5 se ve el formulario de creación de nuevos usuarios.

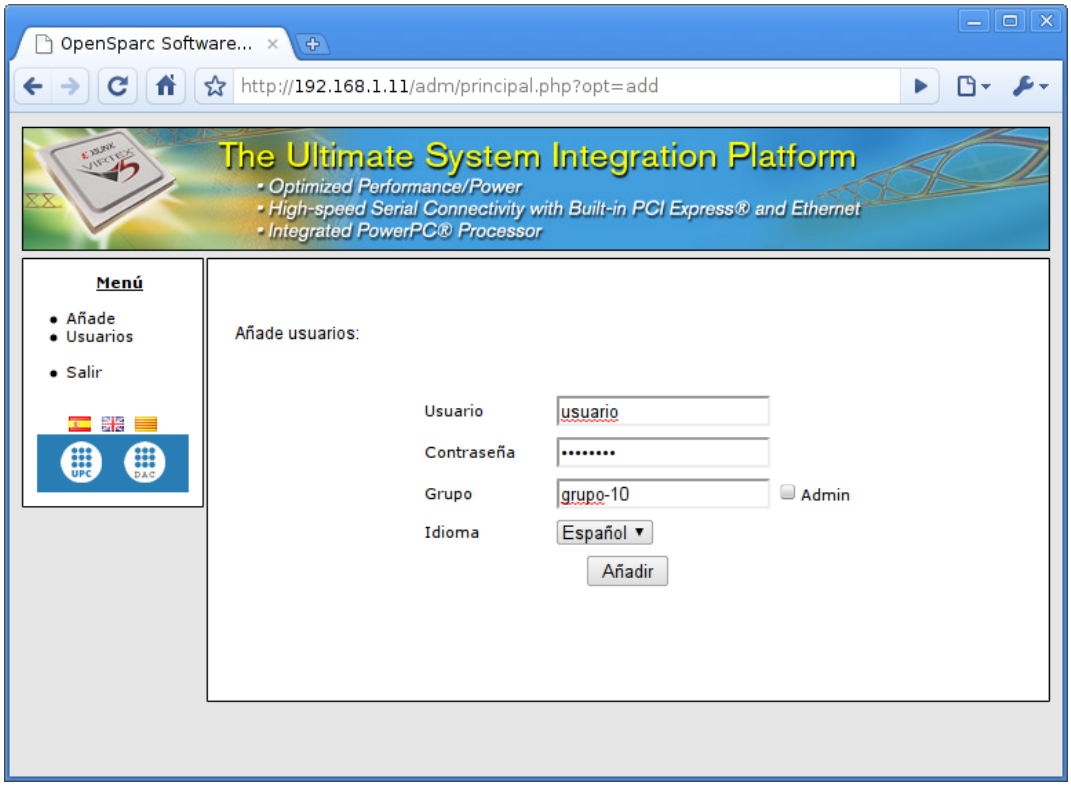

Figura 6.5: *Creación de usuarios en el portal web*

La figura 6.6 muestra los usuarios que se encuentran creados en el sistema actualmente, también proporciona la opción de modificación y eliminación de usuarios.

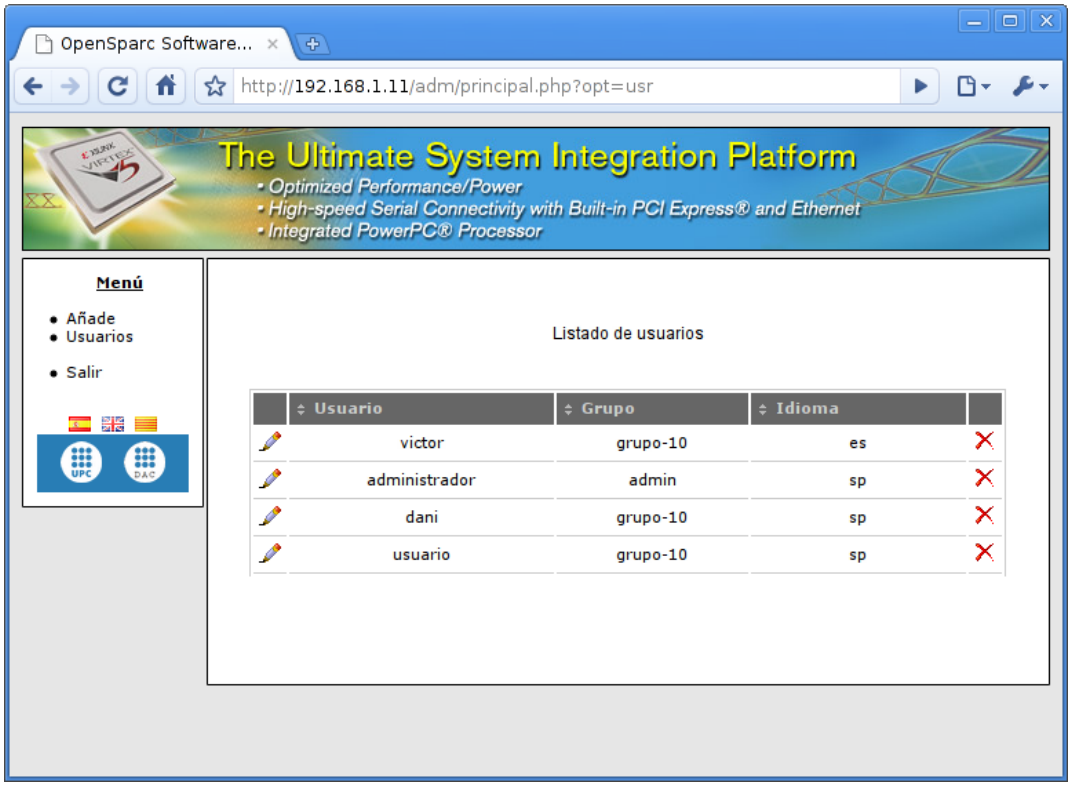

Figura 6.6: *Listado de usuarios en el portal web*

7. LiveCD

Un *LiveCD* es como su nombre indica un *CD* que puede botar por sí mismo un *Sistema Operativo*, totalmente preparado. No sólo sirve para poder transportar tus herramientas, sino que se puede usar para instalarlo y tenerlo todo configurado y adaptado en un simple paso.

En este proyecto también se ha creado un *LiveCD*, que viene preparado con las herramientas necesarias para poder trabajar junto con *OpenSPARC* y *Xilinx*. Viene preconfigurado con todo aquello necesario para que una vez instalado el software propietario de *Xilinx*, funcione todo perfectamente. Este *LiveCD* permite ser usado para desarrollar código por una persona o para ser usado como servidor, donde se recibirán códigos de diversos usuarios y serán mostrados los resultados mediante el portal web del que dispone.

Actualmente crear un *LiveCD* es un tema bastante sencillo, ya que existen multitud de herramientas que permiten hacerlo de una manera rápida y sencilla, ofreciendo unos resultados sorprendentes. La generación de un *LiveCD* no sólo se encuentra disponible bajo sistemas GNU/Linux, sino que también podemos encontrarlo bajo muchos otros como *Free/Open/NetBSD*, **NIX* e incluso bajo entornos *Microsoft Windows*.

Aún teniendo tanta diversidad de plataformas y arquitecturas en el mercado, el proyecto está centrado bajo sistemas *GNU/Linux* y plataformas *i386*, debido a que no es necesario pagar licencias por el uso del *Sistema Operativo* y porque es la arquitectura más común actualmente en el mercado, lo que permite tener un *LiveCD* preparado para la gran mayoría de usuarios.

En el mercado actualmente el número de distribuciones de *GNU/Linux* existentes es muy amplio, tal y como se puede observar en http://distrowatch.com/, donde se recopila el listado existente más grande. Así que el tener que escoger una distribución y adaptarla a las necesidades podría ser una tarea bastante compleja a primera vista. Debido a las razones que se explican a continuación se ha elegido *Kubuntu* como

distribución para el *LiveCD*:

- Soporte: *Ubuntu* es una distribución relativamente nueva en comparación con muchas otras, pero actualmente es quizás la más conocida y con una gran comunidad detrás, que ayuda a resolver muchísimos de los problemas en un tiempo récord, ya que se encuentra todo centralizado en los foros de *Ubuntu*.
- Test: La opción de haber probado esta distribución durante el proyecto, evita el tener que buscar si tiene algún problema de incompatibilidades con la distribución que se podría haber elegido.

Kubuntu de por si es un *LiveCD* bastante completo, ya que contiene el software básico para el trabajo que un usuario pueda desempeñar, a parte de incluir alguna que otra herramienta extra.

La nueva imagen de *Kubuntu*, consta de gran parte de las aplicaciones necesarias para poder trabajar con *OpenSPARC* y *Xilinx*. La única herramienta de la que no consta y que es la base para que todo lo que contiene este *LiveCD* sea completamente funcional y valido, son las herramientas de *Xilinx*, tanto el *IDE* como las herramientas de compilación. El paquete de *Xilinx* no está incluido en el *LiveCD* debido principalmente al tema de licencias y también al espacio que ocuparía, ya que de ser incluidas el tamaño de la imagen obtenida superaría con creces el tamaño de un *DVD*.

A continuación se detallan todas aquellas herramientas nuevas o que sean interesantes para el desarrollo de este proyecto y que están contenidas en el *LiveCD*:

- *Cross-compiler*: compilador cruzado que permite compilar programas desde una arquitectura *x86* hacia una *SPARC*, que es la que viene definida en este proyecto.
- Servidor Web: consta de *apache/php5/mysql* y un portal web que permite el envío de proyectos para que estos sean compilados, ejecutados y muestre el resultado de estos.
- Compiladores varios: contiene compiladores basados en la arquitectura *i386*, para la compilación de programas como el *cross-compiler*.
- *Scripts*: varios *scripts* que permiten la compilación, ejecución y envío del proyecto a la placa para obtener un resultado todo de manera automatizada.

A continuación, se explicará cómo modificar el *LiveCD* existente de *Kubuntu* para este proyecto, haciendo una mención especial en la instalación/configuración de aquello necesario para la plataforma. El proceso de extracción, compresión y generación de la imagen está perfectamente detallado en la web [4], por lo que la explicación no será tan detallada.

El primer paso es bajarse la versión de *Kubuntu* desde la página web oficial [7], la versión con la que se ha trabajado es la *9.04*.

Con los pasos anteriormente descritos, se crea una carpeta llamada *live* en la *home* del usuario. De aquí en adelante todo lo referente al *LiveCD* se hará dentro de la carpeta *live*.

```
1 mkdir ~/live
2 mv ubuntu-9.04-desktop-i386.iso ~/live
3 cd ~/live
```

Seguidamente se montará la imagen en la carpeta *mnt*. Éste es el primer paso para empezar a trabajar con ella.

```
1 mkdir mnt
2 sudo mount -o loop ubuntu-9.04-desktop-i386.iso mnt
```

Ahora se extrae el contenido del existente *LiveCD* en la carpeta creada *extract-cd*.

```
1 mkdir -p mnt/extract-cd
2 rsync --exclude=/casper/filesystem.squashfs -a mnt/
     extract-cd
```

Se extrae el sistema de ficheros *squashfs*. Este sistema de ficheros es el encargado de mantener la estructura de datos que irá en el CD. Y se mueve a la carpeta *edit*, que es desde donde se editará la imagen a nuestras necesidades.

```
1 sudo unsquashfs mnt/casper/filesystem.squashfs
2 sudo mv squashfs-root edit
```

Las siguientes dos líneas copian un par de ficheros para permitir la conectividad una vez se entre en el ambiente *chroot*.

```
1 sudo cp /etc/resolv.conf edit/etc/
2 sudo cp /etc/hosts edit/etc/
```

Finalmente se monta el sistema de dispositivos en *edit/dev*, se entra en un entorno de *chroot*, seguidamente se montan los sistemas de archivos, */proc /sys*, */dev/pts*, para trabajar en la máquina como si se estuviera en local.

```
1 sudo mount --bind /dev/ edit/dev
2 sudo chroot edit
3 mount -t proc none /proc
4 mount -t sysfs none /sys
5 mount -t devpts none /dev/pts
```

Se define la variable *$HOME* y los locales para tener una codificación.

```
1 export HOME=/root
2 export LC_ALL=C
```

A continuación se detallan los pasos para empezar a instalar todos aquellos programas necesarios para que funcione la plataforma y queden integrados entre sí.

El *cross-compiler*, se instalaría tal y como se detalla en el apartado 5.3, con la recomendación de instalar primero el siguiente paquete, ya que es una recomendación básica para su correcto funcionamiento.

```
1 aptitude install texinfo
```

A continuación se definirá el usuario por defecto ya que el sistema de *scripts* funciona mediante este usuario. El usuario que se crea es *sparc*, aunque se puede utilizar cualquier otro. En su *home* albergará una estructura de varios ficheros y carpetas, donde se encuentran los *scripts* que se generan durante la ejecución del *script* principal para la compilación y el lugar temporal donde se compilará todo aquello necesario para *OpenSPARC*.

```
1 sudo adduser sparc
2 sudo gpassw -a sparc scanner,users,lp,www-data
```

Como se observa el usuario *sparc* ha sido creado y añadido a varios grupos, *scanner* y *lp* que permiten tener acceso a la placa y *www-data* que le permite el acceso para copiar archivos desde y hacia el directorio web.

Toda la configuración del sistema web se encuentra detallada en profundidad en el apartado 6.1, tanto su instalación como uso.

Como última parte del *LiveCD*, se instalará un servidor *ftp* para que desde el *Sistema Operativo* que se arranca en la *FPGA*, se pueda acceder para copiar archivos desde/hacia la placa. El servidor *ftp* elegido en cuestión es *proftpd*.

Mediante el siguiente comando se instalará el servidor *ftp* y todas las dependencias que necesite.

```
1 sudo aptitude install proftpd
```

Las siguientes líneas muestran parte de la configuración que es necesaria para el servidor *ftp*. En la primera línea se le define una *shell*. Seguidamente en las líneas 2 y 3 se crea la carpeta por defecto, */home/ftp* y se le asignan los permisos. Finalmente, en las dos últimas líneas, se crea el usuario mediante el cual se podrá acceder vía *ftp* y sin acceso a la *shell* del sistema.

```
1 echo "/bin/false" >> /etc/shells
2 mkdir /home/ftp
3 sudo chmod 777 /home/ftp
4 sudo useradd userftp -p your_password -d /home/ftp -s /bin
     /false
5 sudo passwd userftp
```

Añadiendo la siguiente cláusula dentro del fichero de configuración */etc/proftpd/proftpd.conf* en la sección *Directory /home/ftp*, se configura el servidor *ftp* para dar acceso al usuario *userftp*.

```
<Limit ALL>
        Order Allow,Deny
        AllowUser userftp
        Deny ALL
</Limit>
```

Ahora ya estaría la imagen lista con todos los archivos y configuraciones necesarias, por lo que a continuación se procederá a comprimir nuevamente la imagen.

Las siguientes líneas borran todo rastro de lo que se ha hecho con tal de dejar una imagen limpia al usuario que vaya a utilizar el *LiveCD*.

```
aptitude clean
rm -rf /tmp/* ~/.bash_history
rm /etc/resolv.conf
rm /etc/hosts
```

Acto seguido se desmontan las particiones previamente montadas y se sale del entorno *chroot*.

```
umount /proc
umount /sys
umount /dev/pts
exit
sudo umount edit/dev
```

A continuación se regenera el listado de paquetes que contendrá la nueva imagen.

```
chmod +w extract-cd/casper/filesystem.manifest
sudo chroot edit dpkg-query -W --showformat='${Package} ${
   Version}\n' > extract-cd/casper/filesystem.manifest
sudo cp extract-cd/casper/filesystem.manifest extract-cd/
   casper/filesystem.manifest-desktop
sudo sed -i '/ubiquity/d' extract-cd/casper/filesystem.
   manifest-desktop
sudo sed -i '/casper/d' extract-cd/casper/filesystem.
   manifest-desktop
```

Se genera de nuevo el sistema de ficheros, ya que ha sido modificado y así se actualiza con los nuevos cambios.

```
1 sudo rm extract-cd/casper/filesystem.squashfs
2 sudo mksquashfs edit extract-cd/casper/filesystem.squashfs
```

Como último paso, se elimina el fichero de comprobación *md5sum.txt* y se vuelve a generar tal y como se observa en las líneas 2 y 3. Finalmente se comprime la imagen en el directorio anterior y es llamada *miNuevaIso.iso*

```
1 cd extract-cd
2 sudo rm md5sum.txt
3 find -type f -print0 | sudo xargs -0 md5sum | grep -v
     isolinux/boot.cat | sudo tee md5sum.txt
4 sudo mkisofs -D -r -V "$IMAGE_NAME" -cache-inodes -J -l -b
      isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-
     boot -boot-load-size 4 -boot-info-table -o ../miNuevaIso
     .iso .
```

Con todo esto ya estaría generada la imagen y lista para utilizar. Siempre es recomendable antes de distribuir la imagen comprobar que todo funciona correctamente, esto se puede hacer mediante *VirtualBox*, *VmWare* o cualquier otro software de emulación.

8. Planificación y costes

8.1. Planificación

Este proyecto comenzó a gestarse sobre el 27 de enero del 2009, buscando información referente sobre cómo arrancar un *core* sobre una *FPGA*. Principalmente se debatía entre que *core* utilizar, se tenía en mente *Leon* y *OpenSPARC*, ya que son dos de los más conocidos, además de ser procesadores completos, aunque no se descartaba la posibilidad de que fuera otro. Finalmente a mediados de marzo se decidió utilizar *OpenSPARC* ya que era totalmente compatible con la *FPGA* de la que se disponía.

La planificación inicial constaba de invertir unas 2 o 3 horas diarias, pero esto al final no fue la realidad, ya que algunos días se invertían más horas de las programadas y otros no se llegaba.

El proyecto se basa en varias etapas que eran necesarias ir superando paso a paso, ya que una sin la otra podía ocasionar problemas posteriormente. A grandes rasgos estas etapas son:

- Recabar información
- Conseguir las herramientas adecuadas
- Comprobar y adaptar la *FPGA* a nuestras necesidades
- Generación del *netlist*
- Generación del *bitstream*
- Comprobación de que todo ha ido correctamente, arrancando *GNU/Linux*
- Modificación de la versión de *GNU/Linux*
- Creación del *LiveCD*
- Creación de los *scripts*

- Desarrollo de la web

El primer punto importante era recabar información, ya que era un tema totalmente nuevo para mí y era completamente necesario entender cómo funcionaban todas las partes por separado, lo que requería comenzar a documentarse desde cero. Para ello era necesario resolver muchas preguntas tales como, ¿qué es una *FPGA* y cómo funciona? ¿En qué se diferencia el código *HDL* de código como *c++*? ¿qué es un *netlist* y un *bitstream*? Éstas entre otras muchas preguntas.

Finalmente una vez se tenía toda la información necesaria, se necesitaban las herramientas para poder trabajar. Estas herramientas eran dos principalmente, la *FPGA*, concretamente una *Virtex-5 ML505*, y todas las herramientas de compilación que proporciona la *FPGA*, en este caso las *Xilinx Studio ISE y EDK*, que vienen relacionadas con la placa debido a que hay una estrecha relación entre ambas.

El resto de pasos era cuestión de ir consiguiéndolos uno a uno. Muchas veces algunos se retrasaban mucho debido a pequeños imprevistos que impedían la continuación, tales como la falta de un cable de conexión, el no reconocimiento de la conexión entre el *host* anfitrión y la placa, falta de licencias, el que haya caducado la licencia debido a una actualización del software, etc. Otros pasos como la generación de los *scripts* o el diseño de la web, fueron mucho más rápido de lo esperado.

GANTT project

Nombre	Fecha de inicio	Fecha de fin	Duración
Recabar información	25/03/09	10/04/09	12
Conseguir herramientas	6/04/09	18/04/09	10
Adaptar la FPGA	20/04/09	25/04/09	5
Generación del Netlist	27/04/09	13/06/09	35
Generación del Bitstream	12/06/09	24/07/09	30
Arrancar GNU/Linux	1/09/09	31/10/09	44
Modificar GNU/Linux	2/11/09	7/11/09	5
Creación del LiveCD	9/11/09	15/12/09	26
Creación de los scripts	15/12/09	16/01/10	24
Web	18/01/10	20/02/10	25

Figura 8.1: *Diagrama de Gantt del proyecto*

Como se puede observar, el proyecto ha durado unos 10 meses aproximadamente. Se ha trabajado durante unas 2 o 3 horas diarias, lo que dan un computo global de unas 600 horas. El proyecto para la *ETIS* es de 22.5 créditos, lo que supondría un total de 450 horas. Esto supone un exceso de unas 150 horas, que gran parte

de ellas se pueden justificar debido a que generar un *bitstream* es proceso bastante costoso en cuanto a tiempo, parte del resto de horas se justifican en que un proyecto de este tamaño no siempre coincide con el número de horas previstas, siempre hay problemas que van surgiendo a la hora de realizarlo, como han sido en este caso.

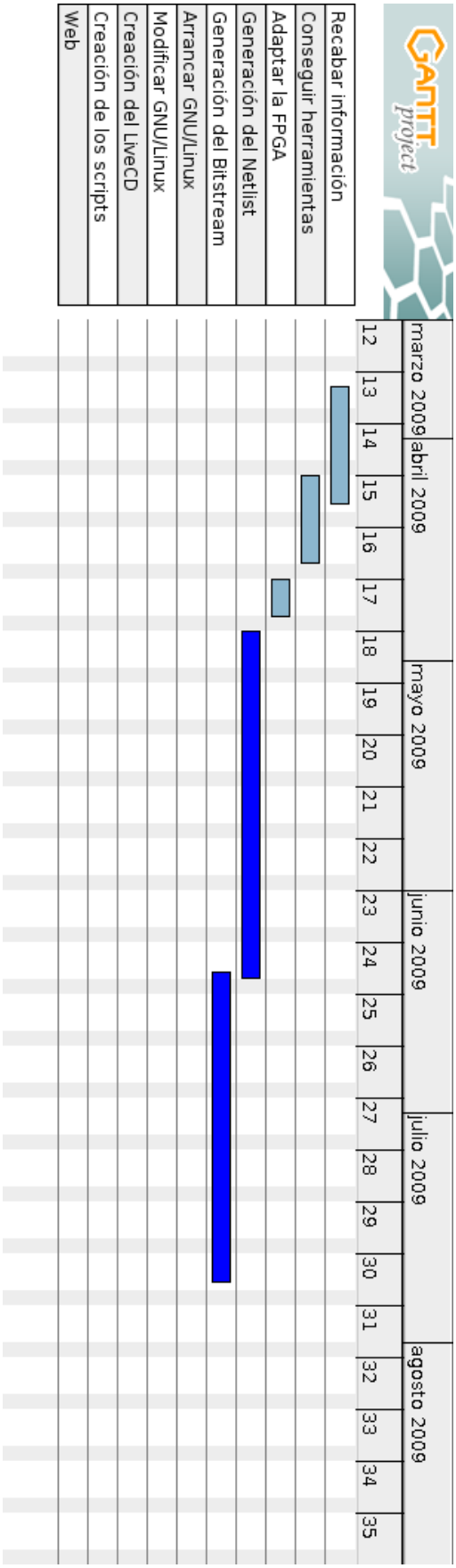

Figura 8.2: *Diagrama de Gantt completo*

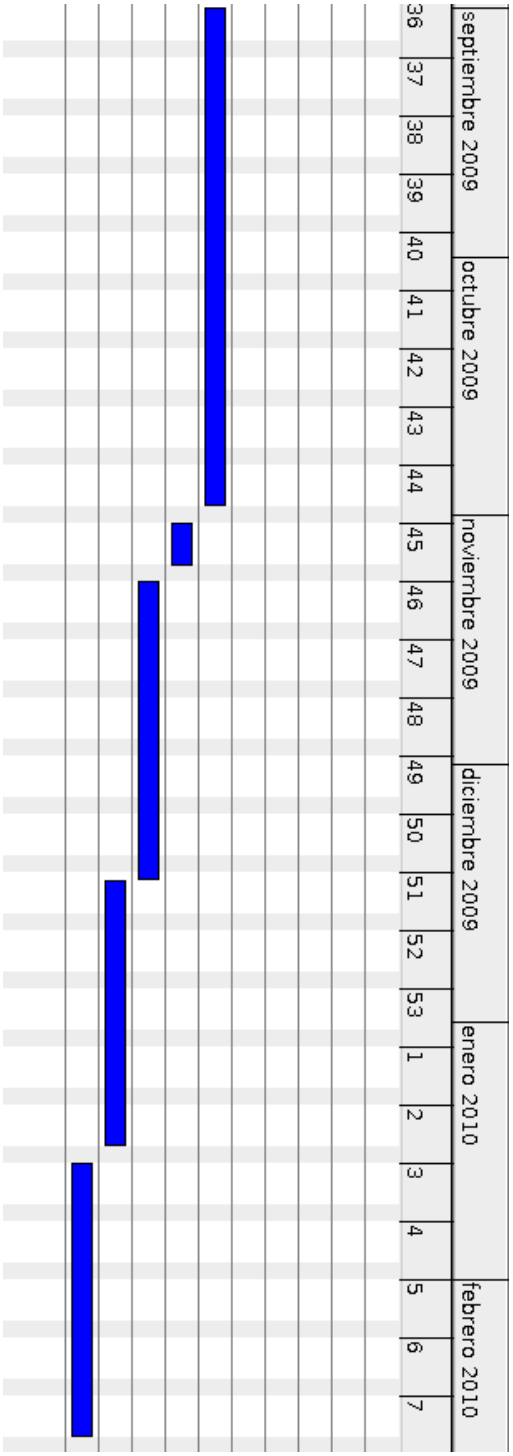

Figura 8.3: *Diagrama de Gantt completo*

8.2. Costes

Como resultado de este proyecto, se pueden hacer unos cálculos aproximados, tanto en el material que se ha necesitado como en el nombre de personas que deberían haber intervenido para la generación de este proyecto, ya que toca varios ámbitos en el campo de la informática y con diferentes cualificaciones.

CiFPGA necesita de varios tipos de informático, debido a que abarca varios ámbitos, desde desarrollador para la plataforma web, gente encargada de sistemas para poder diseñar y programar scripts, profundo conocimiento del *Sistema Operativo*, *GNU/Linux*, para que corra perfectamente sobre una *FPGA*, hasta un becario que se encargue del conexionado entre el *host anfitrión* y la *FPGA*.

Todo los roles que han intervenido son:

1. Becario de sistemas: encargado del conexionado entre la placa y el *host anfitrión*, comprobar el correcto funcionamiento de la *FPGA* y adaptar los jumpers para que funcione correctamente.

2. Técnico de sistemas: encargado de hacer las cosas más triviales como son la instalación del software proporcionado por *Xilinx*, comprobar el correcto funcionamiento de la distribución *GNU/Linux* arrancada en la placa y de la creación del *LiveCD*.

3. Administrador de sistemas: requiere la parte más compleja que es el aprendizaje y correcto uso de las herramientas proporcionadas por *Xilinx* para poder generar el *bitstream*, modificar la distribución de *Ubuntu* y finalmente instalar los drivers para permitir la conexión con la *FPGA*.

4. Desarrollador web: persona encargada de diseñar y programar la plataforma web, que mostrará los resultados de cada ejecución y permitirá el envío de nuevo códigos para ser testeados.

5. Diseñador web: es el encargado de diseñar la estética de la web.

Como se puede observar, cada persona desempeña una función muy específica, de acorde con sus capacidades, aunque durante este proyecto ha sido desarrollado por una sola persona. En un proyecto donde se requieran tantos roles diferentes, haría falta que trabajasen juntos, ya que muchas de las cosas que hace uno, repercuten al resto, como podría ser el caso del administrador de sistemas a la hora de

generar la salida de los *scripts* que serán leídos a posterior por la página web. Aquí el desarrollador web debe ponerse de acuerdo con el administrador para que las salidas funcionen de acuerdo con aquello establecido.

Una vez definida la faena que ha desempeñado cada rol, se puede proceder a calcular el coste que supondría tenerlos trabajando durante el número de horas que se les ha requerido. En la siguiente tabla se puede observar una relación del rol, horas trabajadas, precio por hora y el coste total.

Rol	Precio/Hora	Horas	Precio total
Jefe de proyecto	50 €	20 horas	1.000 €
Administrador de sistemas	30 €	250 horas	7.500 €
Técnico de sistemas	20 €	190 horas	380 €
Becario de sistemas	7 €	30 horas	210 €
Desarrollador web	30 €	100 horas	3.000€
Diseñador web	20 €	10 horas	200 €
Total			12.290 €

A continuación vamos a tratar el coste del material físico utilizado durante el proyecto, esto incluye ordenadores, placas, adaptadores y licencias.

Hardware	Precio
Ordenador de sobremesa	1.000 €
Virtex-5 ML505	900 €
Adaptador USB	10 €
Licencia Xilinx ISE	2.700 €
Licencia Xilinx EDK	500 €
Total	5.110 €

Haciendo un resumen total del coste, el proyecto sale por 17.400 € aproximadamente.

9. Conclusiones

Me gustaría comenzar las conclusiones hablando de que se han conseguido todos los objetivos propuestos, incluso alguno ha sido ampliado.

Los resultados obtenidos durante el transcurso de este proyecto son los siguientes:

1. Configuración de una *Virtex-5 ML505* junto con el procesador *OpenSPARC*, que ha permitido arrancar un programa *standalone* y el arranque del Sistema Operativo Ubuntu.

2. Creación de un portal web que permite la recepción del código de los usuarios y mostrar el resultado.

3. Scripts de generación del *bitstream* de *OpenSPARC* que permiten ser ejecutados manualmente o funcionar bajo las órdenes del portal web.

4. Modificación del sistema operativo que trae por defecto *OpenSPARC* para adaptarlo a nuevas necesidades.

5. Creación de una herramienta que permite la compilación de un código apto para cuando se arranque *Ubuntu* sobre *OpenSPARC*.

6. Servidor *ftp* que permite el envío y recepción de archivos entre la *FPGA* y el *host* anfitrión.

7. Imagen ISO que contiene todo lo referente al proyecto para poder usar todas estas herramientas sin la necesidad de instalarlas.

Las horas dedicadas al proyecto han superado las horas iniciales previstas. Creo que es positivo, debido a que el número de horas no importa cuando a uno le gusta aquello que esta haciendo. Durante este proyecto he disfrutado mucho, ya que no sólo me limitaba a tratar sobre un tema concreto, sino que tenía un amplio conjunto de temas sobre los que trabajar.

Este proyecto ha requerido del conocimiento en varios ámbitos referentes a la informática:

- La administración de sistemas:
  - Instalación de servicios, tales como servidor *FTP*, web, bases de datos.
  - Creación de un *LiveCD* que requería ser modificado para adaptarlo a unas necesidades específicas.
  - Modificación de una distribución recortada, es decir, que apenas dispone de recursos, algunos de ellos básicos, como el gestor de paquetes, algo que hoy en día considero esencial para poder manejar la gran cantidad de programas existentes en *GNU/Linux*, sin contar la gestión de dependencias. Lo que me ha llevado a recordar lo árdua que fué la tarea de instalar programas antes de la aparición del gestor de paquetes.
  - Aprender el funcionamiento de las herramientas proporcionadas por *Xilinx*.
  - Desarrollo de *scripts*, para la automatización del proceso de generación del *netlist*, *bitstream* y obtención de resultados principalmente.
- Conocimientos de hardware: cómo conectar la placa, qué dispositivos son esenciales y cuáles secundarios.
- Desarrollo de software: a la hora de generar la página web.

También me gustaría destacar que mucha de la información no existía en su día, o si la había, ésta era muy pobre. Actualmente, todo esto ha ido avanzando y a día de hoy se pueden encontrar soluciones a muchos de los problemas obtenidos durante el proyecto mediante una sencilla búsqueda en internet. Por suerte, *OpenSPARC* viene con una muy buena documentación, que ayuda a solucionar muchos problemas, otros sin embargo, sólo se encuentran presentes en la web, gracias a que a otra persona le ha sucedido lo mismo y a decidido preguntarlo o simplemente publicar la respuesta, cosas que son de agradecer.

Como valoración final me gustaría recalcar que es un proyecto muy completo, con el que he disfrutado tanto como he sufrido y del que me siento muy orgulloso de los resultados obtenidos.

Bibliografía

[1] *Arquitectura interna de opensparc*, http://opensparc-t1.sunsource.net/specs/OpenSPARCT1_Micro_Arch.pdf.

[2] *Características de la placa ml505 y virtex-5*, http://www.xilinx.com/products/devkits/HW-V5-ML505-UNI-G.htm.

[3] *Características de opensparc*, http://www.mifergo.es/2009/08/opensparc-t1/.

[4] *Cómo modificar un livecd basado en ubuntu*, https://help.ubuntu.com/community/LiveCDCustomization.

[5] *Definición de cpld por wikipedia*, http://es.wikipedia.org/wiki/CPLD.

[6] *Definición de fpga por wikipedia*, http://es.wikipedia.org/wiki/FPGA.

[7] *Descarga de kubuntu desde la página oficial*, http://www.kubuntu.org/getkubuntu.

[8] *Descarga del código fuente opensparc*, http://www.opensparc.net/opensparc-t1/download.html.

[9] *Driver conversor rs-232*, http://www.rmdir.de/~michael/xilinx/.

[10] *Información sobre opensparc*, https://www.opensparc.net/pubs/preszo/OpenStandard.pdf.

[11] *Información varia sobre opensparc*, http://deim.urv.cat/~sun_ca_urv/slides/OpenSPARC.pdf.

[12] *Opensparc dv guide*, "Guía del usuario, incluida con el código de OpenSPARC [8]".

[13] *Opensparc.net*, http://www.opensparc.net/opensparc-t1/index.html.

[14] *Xilinx jtag gnu/linux*, http://www.george-smart.co.uk/wiki/Xilinx_JTAG_Linux#Satisfying_udev.

[15] *Xilinx.com*, http://www.xilinx.com.